UNIVERSITY OF OSLO
Department of informatics

Synthetic Aperture Radar

A Real-Time Processor for ESAs Wavemill Mission

Master thesis

Geir Arild Byberg

May 8, 2012

Abstract

In 2004, the European Space Agency proposed a new Synthetic Aperture Radar mission, Wavemill, which would use new techniques to measure ocean height and ocean velocity down to 10 cm/s. Due to the high sampling frequency and the amount of data that would be produced, on-board processing would be necessary to more efficiently use the communication link, a link that would become a bottleneck if nothing was conducted on the data. An FPGA was recommended for performing the on-board processing.

This thesis focuses on the implementation of a proposed SAR processor that will be used to process the raw SAR data in Wavemill. The SAR processor was synthesized for a Xilinx Virtex-6 FPGA and simulation verified the design for the range compression component. The synthesis showed that implementation of the proposed real-time SAR processor was possible. The synthesis also showed that the SAR processor would function with a clock frequency of minimum 240 MHz.

Preface

This master thesis was carried out at the University of Oslo, Department of Informatics (IFI) in the period February 2011 - December 2011 and at the European Space Technology Center (ESTEC) in the period January 2012 - May 2012. The thesis is for the grade as MSc. in informatics at the University of Oslo and is a 60 point thesis.

I would like to thank my supervisors, Post. Doc. Kyrre Glette and Professor Sverre Holm from Robotics and Intelligent Systems and Digital Signal- and Image Processing from University of Oslo. Thank you for believing in my thesis and for giving me freedom to do what I have always wanted to do. I would also like to give a great thank to Christopher Buck from the European Space Agency (ESA). Thank you for accepting me as a trainee and for providing me with information regarding Wavemill. My best of luck to your project in the future and my hopes that this thesis will be of importance for further development.

A great thanks to Tor-Eivind, Morten, Michael, Martin, Jonas, Martin, Amir, Hans for long chats at the lab, lot of inspiration and making long days to short! A great thank goes also to Olav, Mats, Kristian, Baard and all others who have given me inspiration by their well written thesis' and recommendations to begin early in the writing. I wish I had heard more on that comment!

Last but not least, a great thank to all at ESTEC who have been an inspiration for me, then particular Michael, Martin and the rest of the Wavemill team. May we meet again in the near future!

Geir Arild Byberg
May 8, 2012

List of Acronyms

ASIC	Application Specific Integrated Circuit
ATI	Along-Track Interferometry
BABC	Block Adaptive Bit-rate Control
BAQ	Block Adaptive Quantizer
CSA	Chirp Scaling Algorithm
DFT	Discrete Fourier Transformation
DLR	Deutsches Zentrum fur Luft- und Raumfart e.V.
DSP	Digital Signal Processing
EADS	European Aeronautic Defense and Space Company
ESA	European Space Agency
ESTEC	European Space Resarch and Technology Centre
FFT	Fast Fourier Transformation
FPGA	Field Programmable Gate Array
IDFT	Inverse DFT
IFFT	Inverse Fast Fourier Transformation
IFREMER	French Research Institute for Exploration of the Sea
InSAR	Interferometric SAR
IP	Intellectual Property
JPL	Jet Propulsion Laboratory
LFM	Linear Frequency Modulation
MCU	MicroController Unit
PRF	Pulse Repetition Frequency
PRI	Pulse Repetition Interval
RAR	Real Aperture Radar
RCM	Range Cell Migration
RCMC	Range Cell Migration Correction
RDA	Range-Doppler Algorithm
SAR	Synthetic Aperture Radar
SLC	Single-Look Complex
SNR	Signal-to-Noise Ratio
SPECAN	SPECtral ANalysis
SRC	Secondary Range Compression
UAV	Unmanned Aerial Vehicle
USAF	United States Air Force
WSOA	Wide Swath Ocean Altimeter
XTI	Cross-Track Interferometry

Contents

List of Figures

List of Tables

Chapter 1

Introduction

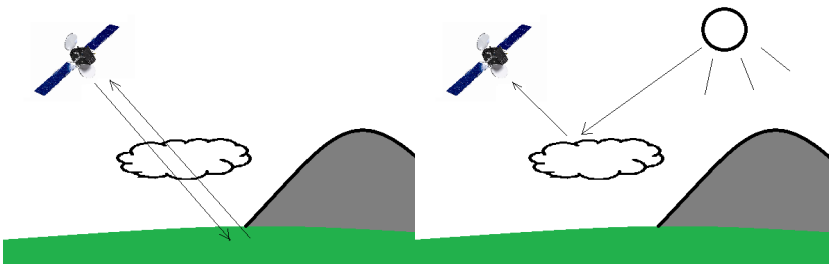
In the last 60 years, Synthetic Aperture Radar (SAR) has been a great contributor to the remote sensing community. Since its first introduction in 1951 [2] and later patent in 1954 [3], it has been used to map the geography and track changes in the surface of not only our but also other planets in the Solar System (Venus, Mars).

With its unique capability to penetrate clouds, illuminate its own area independent of sunlight and provide high-resolution images, it has several advantages when compared to other imaging systems such as optical, a system that cannot see through for instance weather systems and requires illumination from a source, such as the Sun. Figure 1.1 show the difference between an optical imaging system (Figure 1.1b) and an imaging radar system (Figure 1.1a).

In Figure 1.2, the SAR image is located on the left while the optical image is located on the right. The images show the same geographic area but while the optical system only sees clouds and the eye of the hurricane Ike, the SAR system see the surface beneath the clouds.

1.1 Motivation

A high-resolution spaceborne SAR system produces a great amount of raw data. This amount depends on several factors, e.g. dynamic range, spatial resolution



(a) Imaging radar (b) Optical imaging

Figure 1.1: Optical and radar imaging

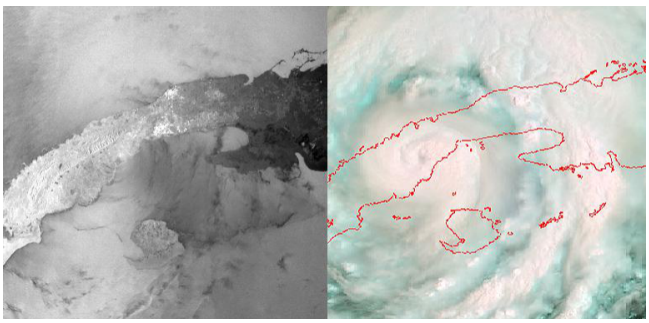


Figure 1.2: SAR image (left) and optical image (right) of Hurricane Ike. Photo: ESA/Envisat

and sampling frequency of the system. The bottleneck in most spaceborne SAR systems today is the communication link between the SAR system and the ground stations. Because of physical limitations, only a certain amount of data can be transferred per orbit. This amount of data depends on the transmit frequency and the diameter of the receive antenna.

Early spaceborne SAR-systems had on-board tape recorders that were used to store the measured signal that were received. When the satellite came in a *line-of-sight* to a ground station, the tape recorder started a playback of its data and transmitted it to the ground station. Modern spaceborne SAR systems collect several raw SAR data products at a time, requiring more storage capabilities and greater communication link than before.

In 2004, the European Space Agency (ESA) proposed a new advanced SAR mission, Wavemill, to monitor and measure the ocean currents and ocean topography, both in the open ocean and the coastal zone [5]. Because of the high-resolution system, one requirement for the mission was to perform real-time processing during orbit flight due to the large amount of data generated by the system. If successful, Wavemill will be the first known civilian satellite where real-time processing is conducted in orbit.

One question that can be asked is why has no one done real-time processing before? In [6], Suess, Schaefer and Zahn try to answer to this. The communication link between a spaceborne SAR satellite and ground stations has never been an issue before, but with higher resolution and larger dynamic range, the amount of data increases and therefore the communication link will become a bottleneck for the system and methods must be developed to overcome this problem. Today, there are two methods that can be considered for this, either increase the bandwidth of the communication link, i.e. increase the data capability of the link, or develop a real-time processor in an attempt to reduce the amount of data.

1.2 Goals and Outline

The overall goal for the thesis is to design a real-time processor for the Wavemill mission that can be synthesized on a single Field Programmable Gate Array (FPGA). The goal can been divided into two main activities: (1) design a real-time SAR processor according to the Wavemill parameters that can be

synthesized on a Xilinx Virtex-6 FPGA and (2) design and simulate at least one compression component that can verify the design of the SAR processor. For this thesis, range compression has been the main component to be designed and simulated.

The outline of the thesis is as follows: Chapter 2 gives an introduction to Wavemill and explains the necessary basics to understanding SAR. In this chapter, a brief overview of on-board processing using FPGAs will be given along with the proposed real-time SAR processor. Chapter 3 presents the implementation and the setup for the simulation of the processor while the results from the different simulations are given in Chapter 4 and the results from the synthesis are given in Chapter 5. Chapter 6 finishes the thesis with a discussion of the results and the final conclusion.

Chapter 2

Background

2.1 Scope

This chapter consists of two parts where the first part introduces the Wavemill mission and the concept of SAR. In the second part of the chapter, an introduction to real-time processing is given. A description of why FPGA has been chosen for the real-time processor is also given. This part includes an overview of previous and current real-time processors. The chapter ends with a presentation of the proposed real-time SAR processor designed in this thesis.

2.2 Wavemill

Ocean circulation is perhaps one of the most important parameters to gain knowledge and understanding of global climate change and modelling of the ocean-atmosphere climate system. With its capability to distribute pollutants and flow of cold polar water, the ocean circulation is a great contributor to the global atmosphere of our planet. Rotation of the earth, winds, temperature, salinity differences and position of the moon and sun all generate ocean currents. A good knowledge of the currents is also extremely important for shipping industry.

Local on site measurements of ocean current velocities and ocean height with buoys have been conducted from the time of the first mariners [7], but since the introduction of spaceborne remote sensing, global ocean current velocities and ocean height measurements can be conducted from space. However, despite its advantage as a global measurement device, spaceborne remote sensing systems can only provide a high-resolution image resolution of a narrow swath [8, 9] to the oceanographers and local in situ measurements are still the best source for attaining good information and fine resolution for the velocity and height of ocean currents.

In the late nineties, NASA's Jet Propulsion Laboratory (JPL) proposed a spaceborne Interferometric SAR (InSAR) mission, called *Wide Swath Ocean Altimeter* (WSOA), whereby new techniques would improve ocean measurements from space [10]. WSOA would use a combination of nadir-looking[1] altimetry and

[1] direction pointing directly below a particular location

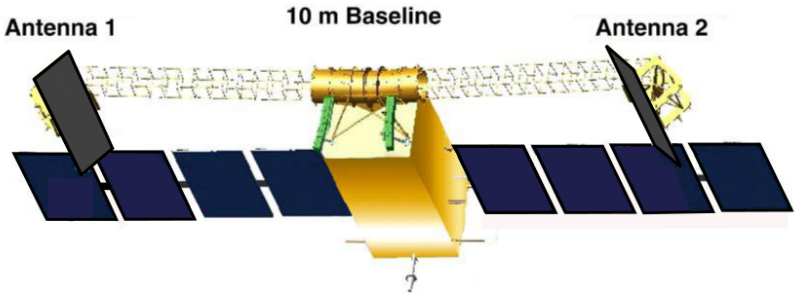


Figure 2.1: The Wide Swath Ocean Altimeter (WSOA). Source: [12]

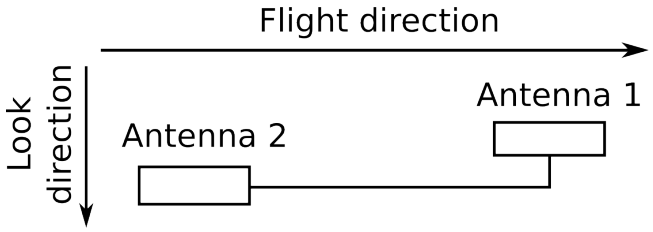


Figure 2.2: Concept design of Wavemill antenna layout

off-nadir radar interferometry to overcome the sampling issues when using regular altimeters. However, the system calibration and knowledge of the baseline attitude and length became an important issue limiting the practical accuracy of the system [11]. Figure 2.1 show an artist's impression of the WSOA. The two antennas are connected by a 10m long baseline and are angled towards each other.

In 2004, the ESA proposed a new ocean current InSAR mission, Wavemill, in an attempt to retain the WSOA benefits while solving the calibration limitations and at the same time conduct both height and movement measurements [13]. Because of the alignment of its antennas (see Figure 2.2), Wavemill can measure both in along-track and across-track directions. This allows direct measurement of 2D ocean surface current by Along-Track Interferometry (ATI) while keeping the ability of WSOA to measure ocean topography by using Cross-Track Interferometry (XTI). The proposed space instrument also aims to perform these measurements with a single pass configuration. Wavemill represents the most ambitious instrument to date regarding the study of the oceans.

2.2.1 Scientific Requirements

A study conducted by Starlab [11] on behalf of ESA showed that the ocean could be separated into two areas, each with different characteristics. The open ocean was defined as the marine environment seaward of the shelf break and the coastal ocean was defined as the part of the world ocean adjacent[2] to the coast and within this point. Figure 2.3 shows the width of the coastal ocean when considering a 200 meter ocean depth.

[2]lying next to

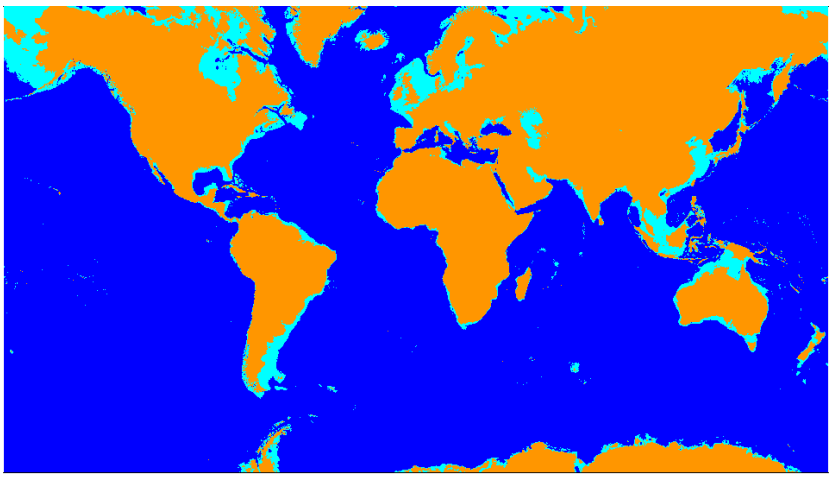


Figure 2.3: Coastal ocean (light blue) and open ocean (blue), 200m isobath. Source: [11]

Phenomenon	**Time scale**	**Length scale**	**Velocity scale**
Equatorial currents	10 days	50 km	10 - 150 cm/s
Western boundary currents	2 days	10 km	10 - 200 cm/s
Ocean mesoscale eddies	5 days	10 km	10 - 50 cm/s
Ocean fronts	5 days	2 km	30 cm/s
Tidal currents	1 hour	0.1 - 20 km	10 - 500 cm/s
Coastal currents	6 hours	5 km	5 - 50 cm/s

Table 2.1: Characteristics of ocean velocities. Source: [16, 17]

Another study conducted by Starlab in association with the French Research Institute for Exploration of the Sea (IFREMER) a time before [11] determined the scientific parameters that Wavemill had to provide information about. Amongst them they were to find the range of observable currents that the instrument should be able to observe, the minimum required resolution of the observation of the currents, the accuracy required and the minimum temporal sampling required for the study of surface currents [14].

The study showed that a minimum requirement for studying the ocean currents were currents with a velocity from 5 cm/s to 5 m/s over a time scale between 1 hour and 10 days, depending on what type of currents was observed (equatorial currents, tidal currents, coastal currents). The different ocean currents and their velocity scale, time scale and length scale is listed in Table 2.1.

A study conducted by the European Aeronautic Defense and Space Company (EADS) [18] on behalf of ESA showed that on-board processing of the SAR raw data would be necessary to reduce the down linked data volume. The processor was required to perform two functions: forming of Single Look Complex[3] (SLC) images from each of the eight antennas and the forming of 4 interferograms[4]. The study recommended the use of one of two SAR processing

[3]SAR processed

[4]Phase difference between two phase-correct SLC images

algorithms, either the unfocused SAR or the SPECAN algorithm. Since a fine slant range resolution was desirable, the SPECAN algorithm was recommended for Wavemill and will be explained in Section 2.3.7.

2.3 Introduction to Synthetic Aperture Radar

A SAR is an instrument capable of obtaining a map of the reflectivity of a planetary surface. By using radar as the main component, SAR has several advantages as briefly mentioned in Chapter 1: it can penetrate through weather systems; operate during both night and day; and can track over a thousand targets at any time.

There is a significant military use of SAR because of its possibility to perform surveillance and targeting. Civilian use of SAR aims more towards environmental monitoring, oceanography and planetary exploration (Venus, Mars). InSAR can provide information regarding changes that have occurred in either topography over a defined time span (hours, days, years) or movements of objects between two SLC images. InSAR depends on true-phase, i.e. the phase is correct after the raw data has been processed.

From a general point of view, a SAR works as follows: The radar transmits a pulse and samples the received signal. Once the radar has sampled the received signal, it moves to a point along the flight direction where it repeats the action. By doing so, the radar synthesizes a long antenna by spending time and a small antenna rather than using a long antenna to gain a fine resolution in the flight direction. By using the radar as the main component, the SAR can detect and calculate the distance from the SAR to the target it tracks.

2.3.1 Range Resolution

The range resolution, ΔR, is defined as the minimum separation between two points with same reflected energy, and that they can be distinguished as two unique points by the SAR system after being processed [1]. This separation can be found by using Equation 2.1 where ΔR is the range resolution in slant range[5], Δf is the bandwidth of the transmitted pulse $(1/\tau)$ and c is the speed of light. The factor 2 is because of the round-trip the signal must conduct, i.e. the travel back and forth.

$$\begin{aligned} \tau &= \frac{2\Delta R}{c} \\ \Delta R &= \frac{c\tau}{2} \\ \Delta R &= \frac{c}{2\Delta f} \end{aligned} \tag{2.1}$$

Figure 2.4 shows the definition of the range resolution. In Figure 2.4 and Figure 2.5 the two buildings are separated by a distance ΔR in the slant range. According to 2.1, if a pulse is transmitted from an antenna, the length of the pulse will determine the smallest distance between the buildings to be considered

[5]Distance in the line-of-sight

as unique after processing. If the separation is smaller than Equation 2.1, the two reflected pulses will merge into one long pulse and we cannot separate them during SAR processing, as shown in Figure 2.5. If the separation is equal or greater than Equation 2.1, the two reflected pulses will be unique and will be separated after SAR processing, as shown in Figure 2.4.

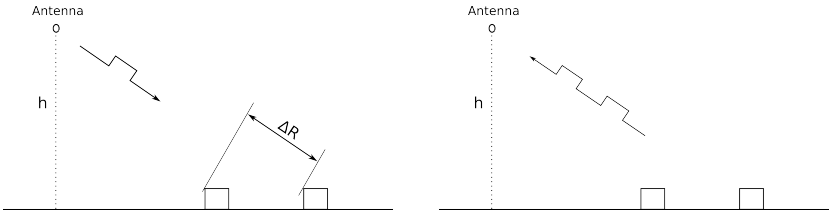


Figure 2.4: Range resolution with good distance between objects

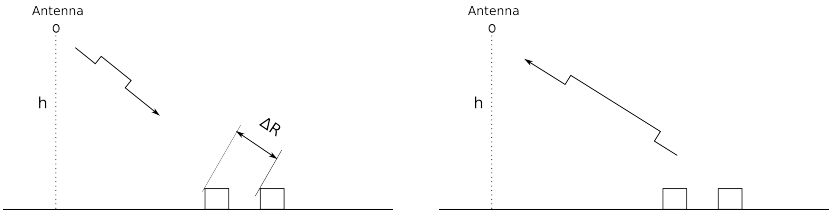


Figure 2.5: Range resolution with too small distance between objects

With Equation 2.1 as the reference for the range resolution, it might be desirable to lower the pulse length to a minimum. However, in SAR processing, this causes new problems. One of them is that the received signal must produce a sufficient echo signal so the Signal-to-Noise Ratio (SNR) can be considered reliable [19, 1].

2.3.2 Azimuth Resolution

To understand how the resolution in the azimuth direction can be found, the definition of the resolution of a Real Aperture Radar (RAR) must first be found [19].

Real Aperture Resolution

The resolution of a RAR, ΔX_{RAR}, is found by using Equation 2.2. The distance between the ground and the antenna is designated by the symbol R, the length of the antenna is designated with the symbol L and the wavelength of the transmitted pulse is designated with the symbol λ.

$$\Delta X_{RAR} \simeq \frac{R\lambda}{L} \tag{2.2}$$

From the equation it can be seen that in order to have a footprint in meters or in centimetres, some of the factors must be decreased to an almost impossible value. Either the range R must be minimized to a few hundred meters, the frequency must be increased to an almost impossible value (to reduce the

wavelength λ) or the length of the antenna must be increased. As an example: Envisat has a wavelength of 6 cm, range is approximately 1000 km and the length of the antenna is 10 m. With these numbers the slant range resolution becomes 6 km.

The electromagnetic spectrum is shown in Figure 2.6. It can be observed that only a portion of the spectrum can penetrate the atmosphere, therefore strongly limiting the possible frequencies possible to use in spaceborne RAR systems. The frequency-bands used today in spaceborne SAR systems lies around 1-10 GHz.

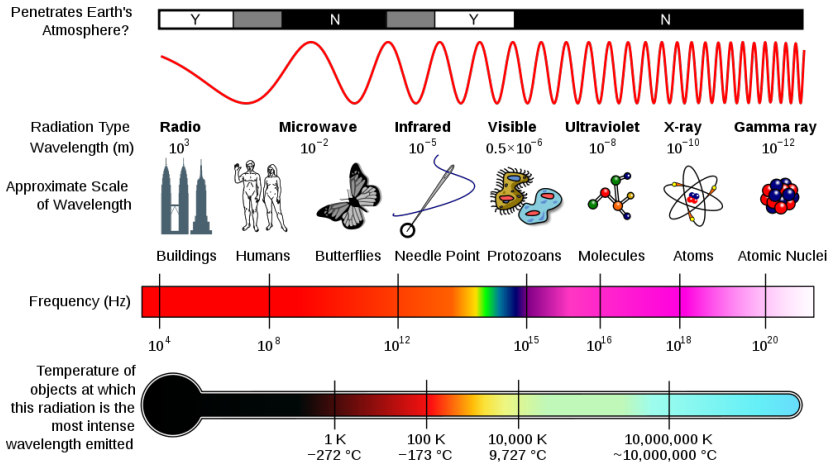


Figure 2.6: The Electromagnetic Spectrum. Source: [20].

Synthetic Aperture Resolution

The most important discovery that led to the development of SAR, was the observation that two point targets with different angles, with respect to the SAR, had different speeds at any fixed position relative to the SAR platform [1]. From this, it was discovered that the different point targets would have two distinctive Doppler frequency shifts. It is this uniqueness that allows the extraction of the synthetic aperture resolution (Equation 2.6).

The Doppler frequency, f_D, at any relative fixed position in the azimuth direction, x, can be described as in Equation 2.3. Note that the position is with respect to the SAR. In the equation, V_r represents the relative speed of the SAR and λ designates the wavelength of the transmitted frequency.

$$f_D = -\frac{2V_r x}{\lambda R} \tag{2.3}$$

Figure 2.7 show the definition of the azimuth resolution. Two objects, T1 and T2 are separated by a distance Δx. The SAR system observes the two targets from a fixed position as according to a reference along its path. The distances from the SAR to the two targets, R1 and R2, are different. From these facts, and from Equation 2.3, the two targets will have different Doppler frequencies from the SAR system's point of view.

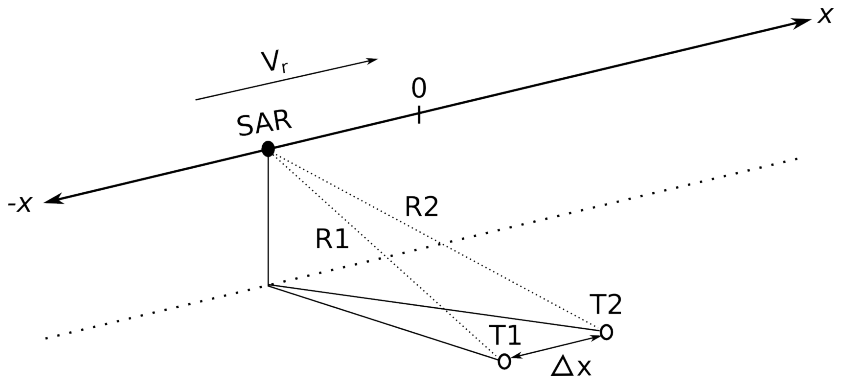


Figure 2.7: Azimuth resolution. Source: [1]

By expanding this equation from a fixed point to the difference between two fixed points, we can use Equation 2.4 to find the Doppler bandwidth. It should also be noticed that when calculating the bandwidth, the result cannot be negative; therefore the sign is shifted and is positive.

$$\Delta f_D = \frac{2V_r \Delta x}{\lambda R} \tag{2.4}$$

The equation for the Doppler bandwidth, Δf_D, can also be written as the inverse of the time span S (Equation 2.5, [1]) of the waveform being analyzed. By setting S to be equal to the Doppler bandwidth and resolving Δx alone on one side of the equal sign, the synthetic aperture resolution can be found (Equation 2.6).

$$S = \frac{R\lambda}{LV_r} \tag{2.5}$$

$$\Delta x = \frac{\lambda R}{2V_r} \frac{LV_r}{R\lambda} = \frac{L}{2} \tag{2.6}$$

This is a very important results. It means that the azimuth resolution is not dependent on the wavelength, pulse length or distance from target, only on the antenna length.

Linear Frequency Modulation

From Equation 2.1 it could be observed that the bandwidth of the transmitted pulse, Δf, determined the resolution in the range direction. However, when designing the physical transmitting system, a high frequency signal with a very narrow bandwidth requires a great amount of energy from the power supply, regardless of the width of the pulse. Therefore, to have fine resolution but still have a feasible design with feasible power consumption, a Linear Frequency Modulated signal (LFM) is used. The LFM signal is a signal that increases the frequency over time, providing a good SNR and reduces the power consumption when compared to a high frequency signal. In SAR processing, the important factor for the range resolution is the bandwidth of the frequency and not the carrier frequency of the system. Figure 2.8 show that the frequency of the LFM

signal increases over time, therefore providing a wide bandwidth and a good resolution.

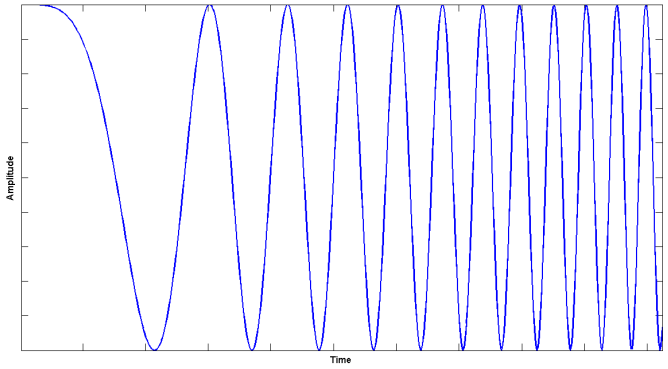


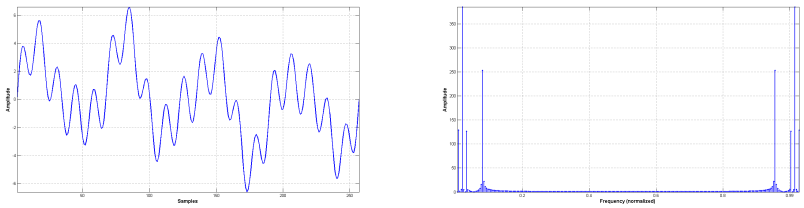
Figure 2.8: Linear Frequency Modulated signal

2.3.3 Pulse Compression

An important operation that must be conducted in SAR processing is the pulse compression of the received signal. It is the compression techniques that allows the range resolution to become as described in Section 2.3.1. Using other words, it is "designed to minimize peak power, maximize SNR, and obtain fine resolution of the sensed object" [19]. In detail, the SAR system performs a linear filtering process of the signal in the frequency domain using three mathematical processes: Discrete Fourier Transformation, matched filtering and inverse Discrete Fourier Transformation.

Discrete Fourier Transformation

Discrete Fourier Transformation (DFT) is a mathematical process whereby a signal is converted from the time domain to the frequency domain. Figure 2.9a shows a signal in the time-domain that consists of three different sine-tones added up and Figure 2.9b shows the energy of the different frequencies in the frequency domain.



(a) Signal in time domain

(b) Signal in frequency domain

Figure 2.9: Signal before and after DFT

The DFT is a summation equation where each input sample, $x[n]$, is multiplied with an exponential factor, as seen in Equation 2.7, to produce the final DFT product. The inverse DFT (IDFT) is shown in Equation 2.8.

$$\text{DFT: } X[k] = \sum_{n=0}^{N-1} x[n] \exp\left\{-j\frac{2\pi kn}{N}\right\} \tag{2.7}$$

$$\text{IDFT: } x[n] = \frac{1}{N}\sum_{k=0}^{N-1} X[k] \exp\left\{j\frac{2\pi kn}{N}\right\} \tag{2.8}$$

The transformations are defined for finite lengths or periodic sampled signals. As can be seen in Equation 2.7 and 2.8, the length of the transformations are limited by the finite parameter N.

Fast Fourier Transformation One disadvantage of using the DFT in a processing system is that it requires N^2 operations to complete the transformation. If N can be factorized into smaller sections, an alternative version of the DFT can be used, the Fast Fourier Transformation (FFT) [19, 21]. By using this transformation instead of the DFT/IDFT, the number of operations will be reduced from N^2 to $Nlog_v(N)$ where N is the power of v. Very often, the FFT is used in processing when N is a power of either 2 or 4 (designated radix-2 and radix-4).

Matched Filter

In SAR processing, a matched filter is a process whereby a known signal is removed from an unknown signal and the output is the energy of the unknown signal without the known signal. In SAR processing this means removing the transmitted pulse from a received signal and the result will be a signal carrying the compressed pulse of the received signal. Equation 2.9 show the concept of matched filter in the time domain. The unknown signal, x[k], is multiplied with the time-reversed filter coefficients, h[n-k].

It should be noted that matched filter is also defined as a "convolution filter" since a convolution process is performed in the time domain. However, in SAR processing, the matched filter is applied in the frequency domain because of the length the matched filter must have in the time domain to process the filtering [19]. From here on, the matched filter will always be defined in the frequency domain if not stated otherwise.

$$y[n] = \sum_{k=-\infty}^{\infty} h[n-k]x[k] \tag{2.9}$$

2.3.4 Range Compression

Range compression consists of the three mathematical processes mentioned earlier. The received signal is first converted from the time domain to the frequency domain by the FFT, then filtered by the matched filter before being transformed back to the time domain by the IFFT. Figure 2.10 shows how range compression is conducted.

Figure 2.10: Range compression

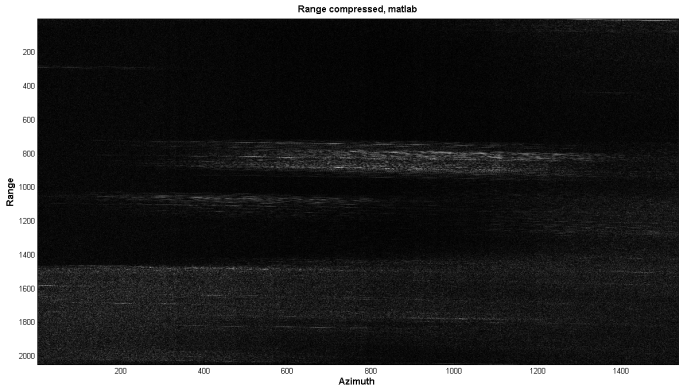


Figure 2.11: Range compressed data

The matched filter is a replica of the transmitted pulse and is defined by Equation 2.10 where f is the carrier frequency of the transmitted pulse (see Section 2.3.2 for information regarding the transmitted pulse), K is the frequency rate in range and T is the pulse length.

$$H(f) = \text{rect}\left[\frac{f}{KT}\right] e^{-j\pi\frac{f^2}{K}} \tag{2.10}$$

2.3.5 Example of Range Compressed SAR Data

Figure 2.11 shows an example of range compressed raw SAR data. After range compressing the raw SAR data, multiple point targets can be observed (the lines in the azimuth direction). To produce a SAR image, another compression must be conducted: the azimuth compression.

2.3.6 Azimuth Compression

Whereas range compression uses the transmitted pulse as a reference when filtering the received signal, azimuth compression is dependent on which processing algorithm has been used to process the raw SAR data. Several algorithms exist to process the data, each with its own advantage and disadvantage. Section 2.3.7 describes the processing algorithm selected for the Wavemill mission, the SPECAN algorithm, and Section 2.3.8 compares it with other processing algorithms.

2.3.7 SPECAN Algorithm

After the successful spaceborne SAR satellites ERS-1 and ERS-2, it was prompted further developments in SAR processing. Could a new SAR processing algorithm be developed so that real-time processing was possible? With such a new algorithm, the SAR system could produce a product with lower resolution than other SAR processing algorithms (Section 2.3.8) [19, 1] for examining surfaces where high resolution could be desired [22] later.

The SPECtral ANalysis algorithm (SPECAN), in its present form, was developed in 1979 by MacDonald Dettwiler and the European Space Technology Center (ESTEC) to design and build a spaceborne real-time SAR processor. The algorithm is efficient and requires less memory than other SAR processing algorithms while at the same time produces an image quality required for moderate-resolution applications [19].

Algorithm Overview

Figure 2.12 shows how the processing of a SAR image by using the SPECAN algorithm is conducted. The SPECAN algorithm uses the same range compression technique as mentioned in Section 2.3.4. After this step a *corner turning* is performed. This is an operation where the range compressed raw SAR data is tilted 90 degrees. The main feature of the SPECAN algorithm is how it conducts the azimuth compression. Instead of performing azimuth compression over the entire azimuth length, the SPECAN algorithm divides the range compressed SAR image into smaller azimuth sections that require less resources than by processing the entire azimuth length.

Range Cell Migration Correction

The first important process after the range compression is the Range Cell Migration Correction (RCMC). After being range compressed, the data gets an undesired feature where the target appears to be migrating over several range cells. This feature is called Range Cell Migration (RCM) and Figure 2.13 show how it occurs. When the SAR passes a target, the slant range[6] from the SAR to the target varies, creating a hyperbolic shape of the received energy. This causes the range information of the target to "migrate" over several range cells as the SAR moves in the azimuth direction. There are several causes to why RCM occurs: rotation of the earth, the width of the antenna aperture (the wider aperture, the more RCM) and the height of the SAR relative to the ground.Since the energy of the target(s) is spread over several cells, the final processed SAR product will have a reduced resolution relative to Equation 2.1.

The RCMC is a processing method that corrects the RCM of the point target from several range cells to only one if the RCMC are performed correctly. In some algorithms, this requires the data to be resampled so the target is located in a correct position. However, doing this is resource demanding for the SAR system and since SPECAN was designed to be a real-time processing algorithm, an easier way to correct the migration had to be developed.

To correct the RCM without using too much resources, the SPECAN algorithm uses a method called linear RCMC. This is a bit different from other

[6] Distance

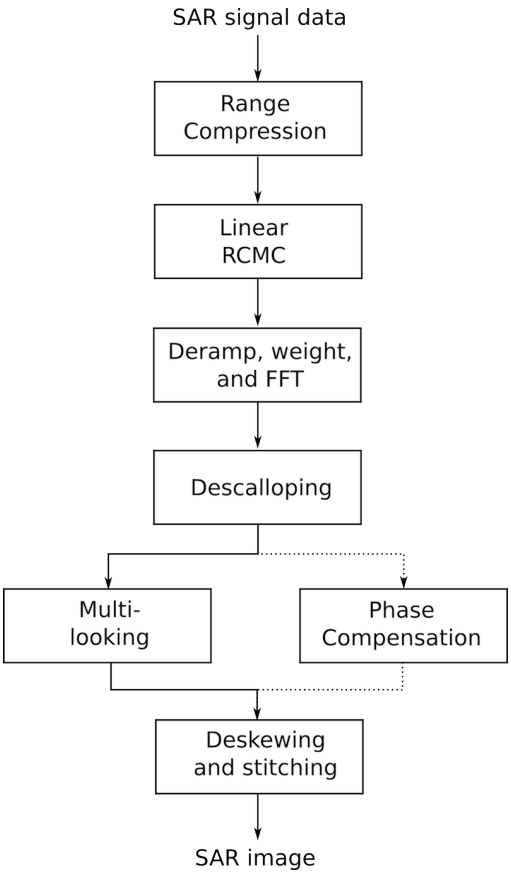


Figure 2.12: Overview of the SPECAN SAR processing algorithm. Source: [19].

SAR processing algorithms that use a hyperbolic RCMC following the range equation [19]. When using a linear RCMC, the range cells are simply "skewed" around a center point to correctly align the RCM of each target. However, for SAR systems with a high *squint* angle, this correction is not considered accurate enough, but for a real-time processing system a trade-off between efficiency and accuracy must made. After the data has been completely processed, i.e. azimuth compressed, the data is deskewed back to its original position so that each point target is located at its correct position.

Figure 2.14 shows how the skewing and deskewing is performed on four range targets. The top left figure shows the SAR signal data after being range compressed. The RCM is linear. The data is skewed in either left or right position, depending on where the center point of the figure has been located. Very often this value is set to be in the center of the SAR data. The top right figure shows the same SAR data after being RCMC. The bottom left figure shows the azimuth compressed target points in skewed position and the deskewed lines. The figure to the bottom right show the SAR processed point targets after the deskewing, now located at the correct position. Figure 2.15 show an example of range compressed data that has been skewed and correctly aligned.

Figure 2.16 show how the received energy and the received frequency appears

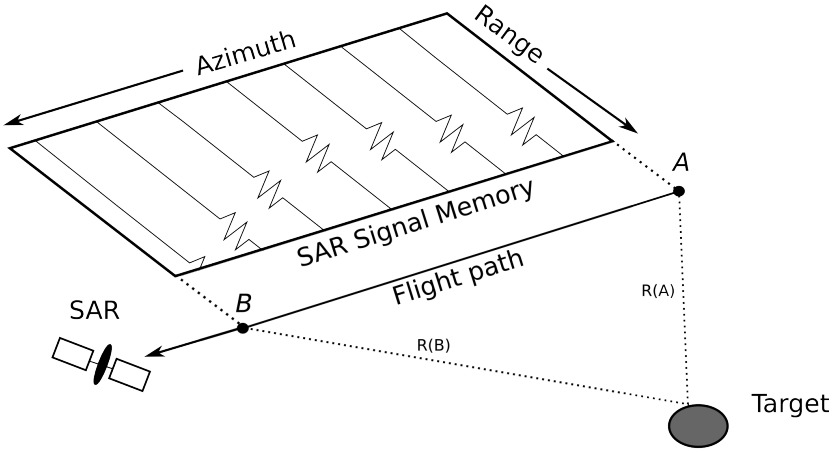


Figure 2.13: Range Cell Migration

of a target as the SAR has moved in the azimuth direction. The distance from the target causes the received signal to generate a Doppler frequency (just as an ambulance that is passing) and the energy has its peak when the target is at is closest. When performing the azimuth compression, the goal is to focus the Doppler frequency to one point.

Deramping and FFT

The deramping process is an operation where the range compressed signal is converted from Doppler frequency to a constant frequency. The deramping process is performed in the time domain and in the azimuth direction, before the FFT is conducted. As mentioned in Section 2.3.7, the azimuth length is divided into smaller sections and because of this, the deramping process can be performed in the time domain. In Figure 2.17 it can be seen how the deramping process is conducted and the effect that it causes. For simplicity, the figure shows the signal after being aliased.

In Figure 2.17A, the frequency component of a target over time can be observed. The signal follows the concept of Doppler history, i.e. the frequency decreases as the SAR passes by it in azimuth time. In Figure 2.17B, the modeled deramping signal after being aliased can be observed. When these two components (A and B) are multiplied, the result becomes Figure 2.17 C, a pure tone over azimuth time.

Pulse Repetition Frequency

As can be seen in Figure 2.17, it is assumed that the frequency component covers an area of one Pulse Repetition Frequency (PRF). PRF is a precalculated value that determines how often a new pulse can be transmitted from the SAR without interfering from other pulses that are being received or reflected. The PRF is found by looking at the time interval between two transmitted pulses,

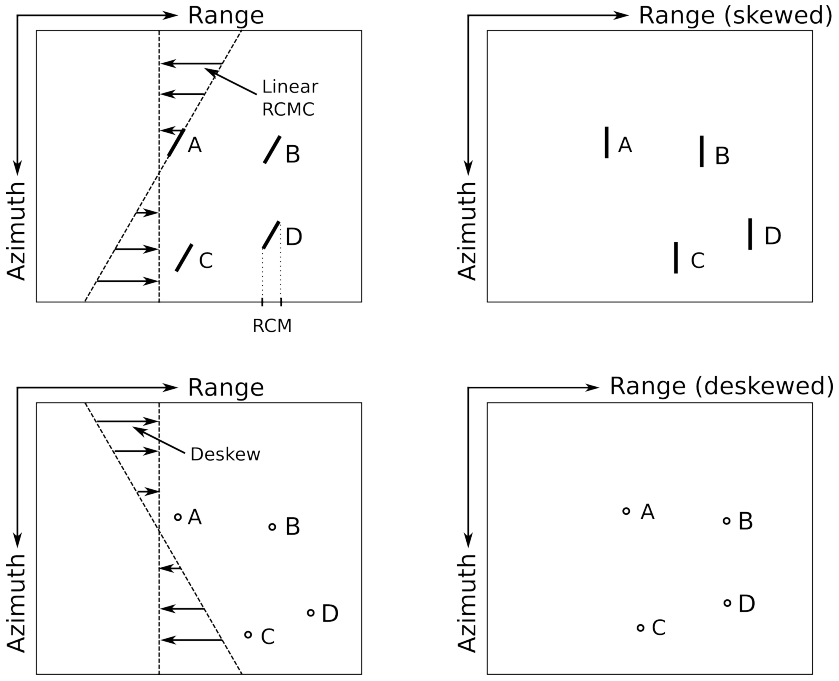
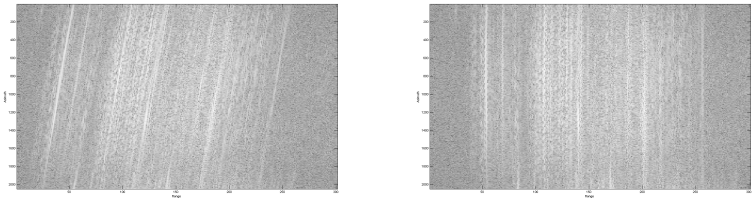


Figure 2.14: Correction of range cell migration. Source: [19]



(a) Before RCMC (b) After RCMC

Figure 2.15: Before and after RCMC

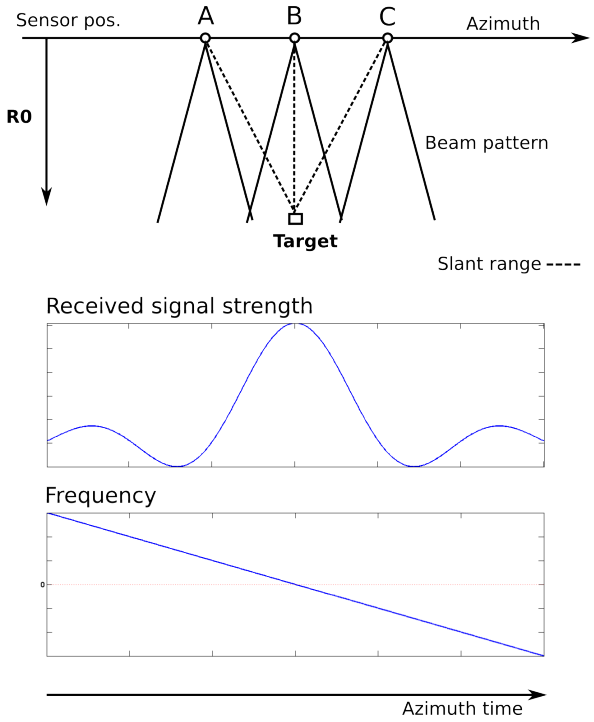


Figure 2.16: Azimuth beam pattern, signal strength and frequency. Source: [19]

the Pulse Repetition Interval (PRI). Figure 2.18 shows the definition of the PRI and Equation 2.11 show how to calculate the PRF from this.

$$\mathrm{PRF} = \frac{1}{\mathrm{PRI}} \tag{2.11}$$

To satisfy Nyquist, i.e. to ensure the Doppler is sufficiently well samples, an expression for PRF is used. Equation 2.12 show the expression where v is the satellite velocity and D is the antenna length. For Envisats ASAR antenna where the antenna has a length of 10 meters and the velocity is roughly 7000 m/sec, the PRF should be over 1400 Hz to satisfy Nyquist.

$$\mathrm{PRF} > \frac{2v}{\mathrm{D}} \tag{2.12}$$

After the deramping operation is completed, the SAR system performs an FFT operation. The main difference between the SPECAN algorithm and the other SAR processing algorithms is that it processes only sections of the azimuth data instead of all samples in the azimuth direction. This drastically reduces the resources needed but also reduces the resolution of the system and this will be discussed in a later chapter.

Descalloping

Since the energy from successive targets is extracted from different parts of the non-uniform Doppler spectrum, the SPECAN algorithm suffers from a radiomet-

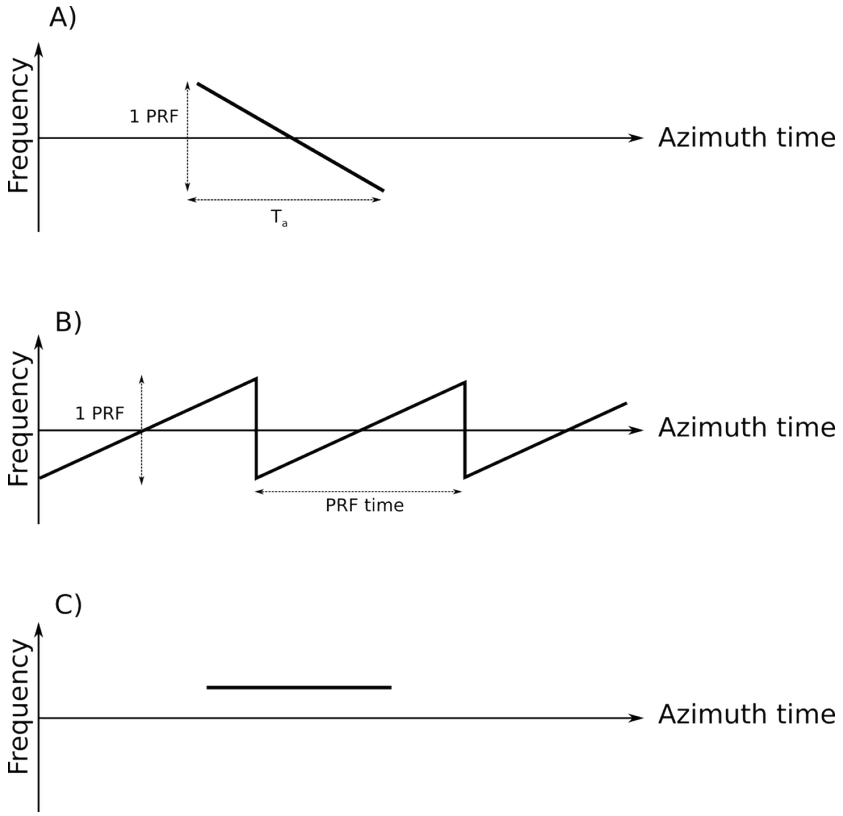


Figure 2.17: The deramping effect in SPECAN processing. Source: [19].

ric periodic variation known as *scalloping*. Using other words, the backscatter from the different targets depends on where in the antenna beam they are situated. If a target is located in the center, its reflected energy will be considered high, at the beam peak. If the target is located a bit off-center, the energy of the target will be lower.

The main purpose of the descalloping is to make sure that the output data has a constant energy. To do this, the signal is simply multiplied with an inverse of the energy profile [19]. If the compensation is done correctly, the result will be constant with time. One requirement for this compensation is that the received energy is a symmetrical function over each FFT cycle. This requires that the Doppler centroid is known correctly.

One way of reducing the scalloping effect is to use *multilook* systems. Mul-

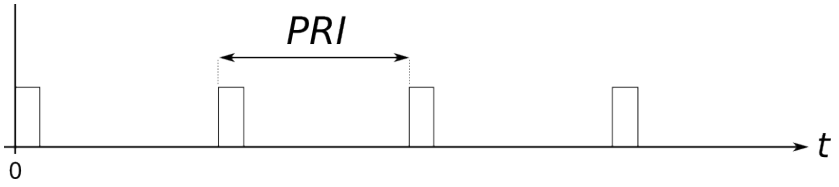


Figure 2.18: Pulse Repetition Interval

tilooking is an operation performed in the SAR system where processed SAR images from different parts of the signal spectrum are averaged. Using a multi-look system also reduces the amount of speckle noise [19, 2] and a higher SNR can be achieved.

Phase Compensation

Since Wavemill is meant to be an InSAR mission, the phase information from the raw data is very important. Therefore, producing a SLC image will not be sufficient. After the range and azimuth compression, the phase of the image must be corrected so that the InSAR product becomes correct. This is done by conjugating the phase terms in the descalloped signal [19] and modify the time variable used in the exponential terms. The goal of the phase compensation is to make sure that the phase of each target has its peak period instead of constant linear growth.

2.3.8 Comparison With Other Algorithms

In [19], Cumming writes about the most common SAR processing algorithms, their advantages and disadvantages. One of the most used and recognized algorithms is the Range-Doppler Algorithm (RDA). It was developed to process the SAR data from RADARSAT-1 and is still in use today. The RDA has a main advantage that it provides high-resolution products to the user and is easy to implement since it performs all calculations in the frequency domain.

Compared with SPECAN, RDA is more feasible to use after transmitting the data to a processing system. This is because the algorithm requires a large amount of processing power and resources. Another disadvantage with the RDA is that the RCMC is nonlinear. Since the RCMC is dependent on the knowledge of the range to the target and the velocity of the SAR system, the RCMC requires too many resources and is therefore unfit for the Wavemill. For an airborne case the velocity can easily be extracted, but for a spaceborne system the velocity is dependent on the range and therefore complicates the processing.

The main component that causes the RDA to be inefficient for Wavemill is the performance of the Secondary Range Compression (SRC) because of the high squint. This compression technique corrects the phase of the SLC image so that a production of an interferometric product is possible. Using this method requires the user to have knowledge about the altitude and the azimuth frequency as well as modifications to the azimuth matched filter. As in the case with the non-linear RCMC, the SRC would require too much of the resources of the overall processing system.

Another algorithm is the Chirp Scaling Algorithm (CSA) [19]. It was developed to eliminate the nonlinear RCMC component in the RDA algorithm. The CSA uses a scaling principle whereby a frequency modulation is applied to a chirp-encoded signal to achieve a shift or a scaling of the signal. The main advantage with CSA is that it causes the SRC to be made dependent on the azimuth frequency, an advantage that allows the processing to be performed in the two dimensional frequency domain. But, as stated in [19], an approximation used in the IFFT operations causes the CSA to be not so adequate for wide apertures or highly squinted systems.

The last SAR processing algorithm mentioned in [19] is the Omega-K algorithm. It is a frequency domain processing algorithm that consists of two main steps: reference function multiplication and Stolt interpolation. The Stolt interpolation performs a mapping of the range frequency axis, which achieves the differential focusing of targets away from this reference. However, the Omega-K algorithm relies on a constant effective radar velocity. To compensate for this, the resulting error must be analysed on a case-by-case basis, especially when high-resolution is needed. This process would require too much time and too many resources from Wavemill to be used.

2.4 On-Board Processing

On-board processing for spaceborne SAR missions is a subject that have widely discussed in the recent years [6, 23, 24, 25, 26]. The definition of on-board processing in this thesis is that the SAR processing is conducted in real-time while the SAR system is in orbit. For Wavemill, a spaceborne SAR mission that contains a high-resolution radar [13], on-board processing must be performed to avoid the bottleneck that occurs in the communication link.

One key component that encourages the use and development of such systems is the usage and possibility to build more advanced sensors, and the information attained from the SAR product can be transmitted to the users faster than before. However, using a real-time processing system will also require a trade-off between the precision and the volume of data produced.

2.4.1 Existing On-Board Processors

The number of active on-board processors in SAR missions is as of today not known. The last known SAR-satellite that had an on-board processor was ESAs ERS-1. It had an real-time on-board range compression using a SAW filter. However, the system was never used. There might be several reasons for this. First, it is more desirable to process the SAR raw data on the ground, with more processing power and resources available. By doing so, the accuracy of the processed SAR product is high and the highest resolution possible from each sample point can be achieved. Second, it is time consuming to design and/or develop a new real-time processor for each new mission, depending on which technology being used, based upon the requirements (hard/soft deadline) etc. Even though there do not exist any on-board processors in satellites, there are some that exist within airplanes.

E-SAR

The German Aerospace Center (DLR)[7] developed an on-board processor for their E-SAR airplane [27] in the 1980s and successfully delivered its first images in 1988 in its basic system configuration. The system is designed so that it can process 8 or 16 MB/sec (selectable), depending on the wordlength (6 or 8 bits, complex), the bandwidth (50 or 100 MHz) and the range and azimuth resolution.

[7] Deutsches Zentrum fur Luft- und Raumfahrt e.V.

In 2006, F-SAR, a follow-up to E-SAR, was developed and made its maiden flight in the same year [28]. The F-SAR continues with the technology of E-SAR and contains several antenna configurations in a higher frequency-band along with significant improvements from E-SAR (increased data rate, increased range coverage, increased sampling).

Environment Canada Convair 580

The Canadian Convair 580 is an advanced airborne SAR system that contains both C- and X-band systems, can provide 6 m resolution and can provide both ATI and XTI. The system provides real-time and post-flight data streams and on-board processing is possible. The Convair was used to prepare for the advanced features of Radarsat-2.

Unmanned Aerial Vehicles

In the more recent years, Unmanned Aerial Vehicles (UAV) built for military purposes have been built with SAR antennas to be used in surveillance and for civilian missions. In 2011, NASAs Dryden Flight Research Center acquired two UAVs (Global Hawk) to develop a new SAR system, UAVSAR, where the SAR is a part of the UAV. For future studies and technology developments, on-board processing will be an important factor for further development.

2.4.2 Using FPGAs as On-Board Processors

FPGAs have high throughput, the user has direct control (can adjust every bit if necessary) and they are power efficient. When comparing FPGA with other similar technology, some differences occur and will be explained here.

A comparison between Application Specific Integrated Circuits (ASIC) and FPGAs conducted in 2007 [29] showed that the average area consumption of the FPGA were 20-30 times greater than an ASIC made by the same technology (90 nm processing technique). However, when more resources were used in the FPGA, this ratio showed a tendency to decrease closer to 10. The same paper also showed that the critical path delay ratio were on average 3.2 times higher in an FPGA than the ASIC. Despite this, the FPGAs main advantage is that it is reprogrammable and FPGAs are perfect candidates for use in real-time systems where firm deadlines exist.

ASICs for space are very power efficient due to the low power consumption. The problem is the development time and the high expense as typically they are designed for a specific mission. ASICs are also preferred for deep space missions as they can be made very radiation hardened.

Another technology that might be considered for use in future SAR missions is MicroController Units (MCU). However, when having a closer look into the design of MCUs, they are based upon ASIC design and their reprogrammability bases itself upon instructions. Another difference between MCU and FPGAs is that while the MCU is considered time expanding (more instructions require more clock cycles but area consumption remains), the FPGA can be considered area expanding (more code requires more area but not more clock cycles).

Using FPGAs as SAR processor is not a new subject. In 2004, researchers at JPL proposed an on-board FPGA-based SAR processing system for future

space borne systems [24]. In their proposal, JPL mentions challenges and the major objectives for an FPGA-based SAR processor. NASA and United States Air Force (USAF) identify on-board processing as needed technology and this stands as the main cause. The proposal showed that using on-board processing using FPGAs in future SAR missions is feasible.

Astrium GmbH has also discussed the subject of on-board processing of SAR data [6]. In 2000, they presented an article where different processing methodologies were mentioned; Block Adaptive Quantization (BAQ); Digital Filtering; and FFT-Block Adaptive Bit-Rate Control (FFT-BABC). Astrium presented the advantages and disadvantages with each of these methodologies. The article also mentions that there are more potential and possibility to develop such a system if it processes off-line (when all the raw data has been stored). By doing so, the demand for resources is reduced and the system can spend more time processing since it is considered non real-time. One important issue that can be mentioned is that the space technology at the time the article was written (year 2000) had lack of resources and possible radiation hardened FPGAs. Since then, Xilinx has introduced new radiation hardened FPGAs, Virtex-4 and Virtex-5.

Chapter 3

Implementation

3.1 Scope

In this chapter, the implementation of a proposed real-time SAR processor for Wavemill will be described in Section 3.2. Section 3.3 will give a description of the software tools used in the implementation, then particular the Xilinx Core Generator. Section 3.4 gives a description of the design and implementation goals that was set to this thesis. Section 3.5 gives an overview of the implemented SAR processor before a detailed description of the implementation of the range compression and azimuth compression component is given in Section 3.6. The chapter ends with a description of the setup of the simulation and the setup for the synthesis, respectively Section 3.7 and Section 3.8.

A total of four range compression components and one azimuth compression component were designed and implemented in the hardware language VHDL for this thesis. Two of the range compression components were designed to verify the functionality of the design by simulation while the other two components were designed as according to the specifications of ESAs Wavemill mission [18] for synthesis. The azimuth compression component were designed and implemented as according to the specifications of ESAs Wavemill missions [18].

3.2 Proposed SAR Processor for Wavemill

The proposed real-time SAR processor for Wavemill can be observed in Figure 3.1. The processor is based on a design from [18]. From the figure, it can be observed that some of the SPECAN algorithm components are not included, amongst them is the linear RCMC, descalloping and the deskewing component. These components have not been designed in this thesis because the proposed SAR processor is a preliminary suggestion. However, for this thesis, the processor will be used so that a minimum requirement for resources and clock frequency can be found before expanding the system.

The proposed SAR processor will be designed towards the specifications that are listed in Table 3.1. From the table, it can be seen that the designed processor must achieve a clock frequency of at least 120 MHz. The input data will have a wordlength of at least 16 bit complex data where the 8 most significant bits

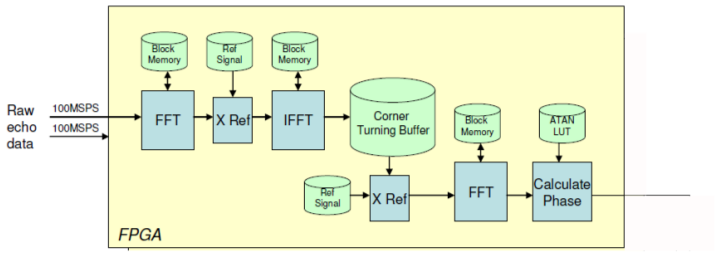


Figure 3.1: Proposed SPECAN Algorithm for Wavemill. Source: [18]

are real and 8 least significant bits are imaginary. Wavemill will load 32,905 complex samples for every PRI and the processing of the azimuth samples will be conducted for every 50 bursts, i.e. for every 50th azimuth sample.

Parameter	**Value**	**Units**
Sampling rate	120	MHz
PRF	2200	Hz
Range samples	32,905	samples
Azimuth samples (burst)	50	samples

Table 3.1: Technical requirements for Wavemill. Source: [18]

3.3 Used Tools

3.3.1 Xilinx Core Generator

The Xilinx Core Generator is software that generates the necessary IP cores that will be used in this thesis. The cores cover a range from the most basic to the more advanced cores within communication, arithmetic and digital signal processing. In the Core Generator, one can decide the optimization of the IP core either in performance or in resources.

When generating a core, the software produces all necessary files that are needed to verify its functionality by simulation and the use of resources when being synthesized (testbench, netlist). However, some of the files that are generated can only be used for simulation purposes.

The Core Generator reduces the design time and enables a fast verification and a good optimization of the overall system, either if it is balanced, optimized for speed or optimized for resources. This is a positive asset from the software

since it will then allow its users to have more time to perfection the overall system.

3.3.2 Other tools

The other tools used in this thesis were MentorWorks' ModelSim SE 10.0d (64-bit), Mathworks MATLAB R2009b (32-bit) and R2011b (64-bit) and Xilinx ISE Design Suite 13.3. The ModelSim software was used to simulate and compile the VHDL code while the Design Suite was used to synthesize the different components and the SAR processor. The MATLAB software was used to extract the dataset (see Section 3.7.2 for more detailed information regarding the dataset) and prepare it for simulation. The same software was also used to generate the expected products that were used to verify the correct functionality of the implemented SAR processor.

3.4 The Architecture Design and Implementation Goals

The architecture that was to be implemented in this thesis was based upon the design that was described in Section 3.2. The main goal of the thesis, as mentioned in Chapter 1, was to find out if a complete real-time SAR processor, designed for ESAs Wavemill, could be synthesized into one FPGA. In this thesis the FPGA would be a Xilinx Virtex-6. An explanation to why this FPGA was chosen will be given in Section 3.8.

The implementation of the real-time SAR processor had to follow the technical requirements for Wavemill, as described in [18]. To be considered successful, the SAR processor that was designed in this thesis had to accomplish the following implementation goals:

- Achieve a clock frequency of at least 120 MHz
- Manage to process at least 32,905 samples in the range direction
- Manage to process at least 50 samples in the azimuth direction

Some modifications of the implementation goals were needed because of the remaining time to complete the thesis. The first modification was to increase the number of processed samples to the nearest value of power of 2. This means the samples in the range direction increased from 32,905 to 65,536. This would reduce some of the complexity of the overall system, mainly by neglecting any control systems that would be needed in a regular SAR system. The second modification was to increase the number of processed samples in the azimuth direction from 50 to 64.

With the new modifications in place, the implementation goals became the following:

- Achieve a clock frequency of at least 120 MHz
- Manage to process at least 65,536 samples in the range direction

- Manage to process at least 64 samples in the azimuth direction

The implementation of the Wavemill SAR processor will be as according to these implementation goals.

3.5 Overview Of The Implemented SAR Processor

A top level schematic of the designed SAR processor can be seen in Figure 3.2. The function of the processor is to perform range and azimuth compression of the received raw SAR data in real-time, i.e. processing the data directly without losing any data, as according to the description given in Chapter 2.

Figure 3.2: Implemented SAR processor, overview

3.6 Implementation Details

A detailed design of the implemented SAR processor can be seen in Figure 3.3. It can be observed that the received signal, *SAR data*, is first range compressed, then corner turned before being azimuth compressed, following the description given in Section 2.3.4. The corner turning has been recommended to be performed on an external component, preferably an ASIC, and has therefore not been implemented in this thesis.

Figure 3.3: Detailed overview of the SAR processor

3.6.1 Range Compression Component

The range compression component was designed with the mathematical functions described in Section 2.3.3. Figure 3.4 show the layout of the implemented design. The input is first converted to the frequency domain by the FFT before being filtered by the matched filter. Finally, the IFFT converts the now filtered and compressed signal back to the time domain.

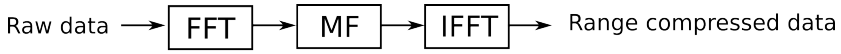


Figure 3.4: Range compression component

FFT

The FFT component was first intended to be hard coded and designed after the principle presented in [21]. The component was based upon a "butterfly" diagram, as shown in Figure 3.5. Several butterflies were connected in serial and parallel to form an N-point FFT structure.

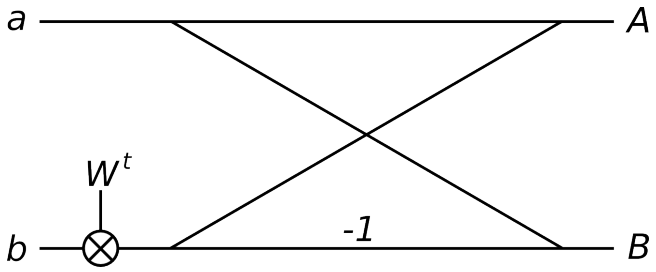


Figure 3.5: Butterfly diagram. The structure performs a 2 point DFT operation of the two inputs. Source: [21]

Equation 3.1 shows the results, A_o and B_o of a 2-point DFT after the two inputs, A_i and B_i, have been transformed. The W^t factor is called the "twiddle factor" and is an exponential function (see Equation 3.2). The value of t can be found in [21].

$$A_o = A_i + B_i W^t$$
$$B_o = A_i - B_i W^t \tag{3.1}$$
$$W^t = e^{j2\pi t/N} \tag{3.2}$$

In an N-point structure, the butterfly diagrams are connected together in a matrix-shape. In parallel, N/2 diagrams are used to perform 2-point DFT processing, while in serial, log2(N) diagrams are connected together to perform the remaining N-point FFT. Figure 3.6 show an example of how the diagrams in an 8-point FFT structure are connected together. It should be noted that the inputs have been bit reversed in order to have a natural order of the outputs.

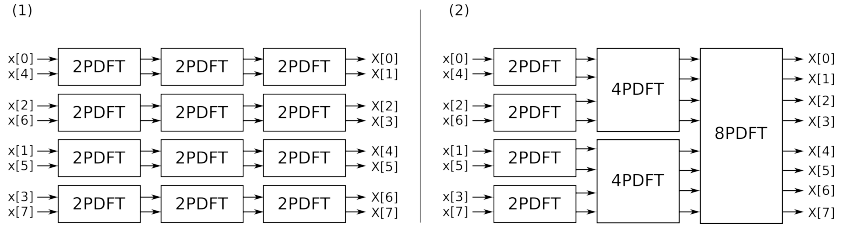


Figure 3.6: 8-point FFT structure. (1) show how the butterfly diagrams are connected together while (2) show what transformation that is being performed. Source: [21]

During the development, the hard coded component was simulated and showed very good results and had a processing time as low as log2(N) clock

cycles. However, when synthesizing the system for the desired FPGA, it was evident that the amount of resources that was needed far exceeded what was possible to attain from it, in some cases as high as 3 times the possible resources available[1]. Therefore, it was determined that the best choice for further development of the processor was to use IP cores from Xilinx. The cores were generated by the Core Generator (described in Section 3.3). Xilinx can provide four different architectures of the FFT component, each with its own feature [31]:

- Pipelined streaming
- Radix-2 Burst
- Radix-2 Burst Lite
- Radix-4 Burst

Pipelined Streaming Architecture The pipelined streaming architecture provides a high throughput[2] but also requires most resources of the FPGA. The advantage by using this architecture is that it simultaneously performs calculations on the data while loading new frames and unloading the results of processed data.

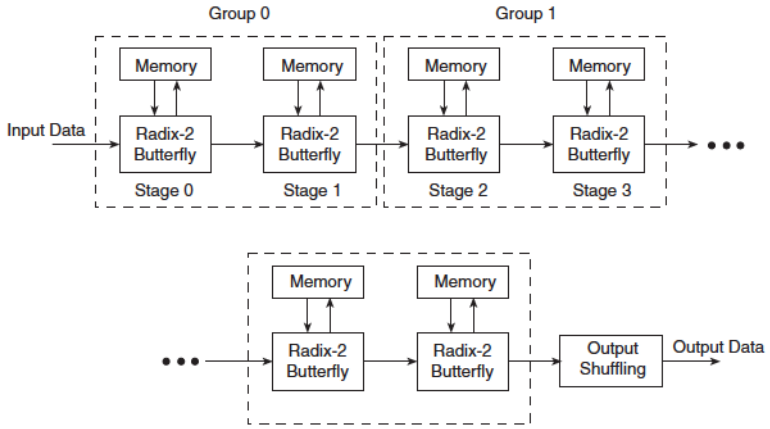


Figure 3.7: Architecture of pipelined streaming FFT. Source: [31]

Figure 3.7 show the design of the pipelined streaming architecture. From the figure, it can be seen that the architecture is built upon multiple serial connected butterfly diagrams (as mentioned earlier), something that requires more resources from the FPGA. It should be noted that the pipelined streaming architecture, as seen in the figure, is defined as a radix-2 streaming architecture, i.e. the transformation length is a power of 2.

Another important feature that might explain why this architecture uses so many resources is the number of memory components. As can be observed in

[1] Synthesized on both Virtex-6 series and latest 7-series from Xilinx
[2] How much data that passes by at a certain point

the figure, each butterfly diagram is connected to a memory component. By having so, the architecture uses a module in the FPGA that is dedicated to behave as a memory component. This will increase the speed but also require more resources. As can be observed, the pipelines streaming architecture uses log2(N) memory components as data and phase storage of the stage calculations. To best take advantage of the resources that the FPGA has, these memory components are BRAM blocks.

Radix-2 Burst While the pipelined streaming architecture is built upon several radix-2 butterflies in serial, the radix-2 burst architecture use only one butterfly processing engine. As can be viewed in Figure 3.8, the radix-2 burst architecture is designed as a "recursive" component, i.e. the output of the butterfly processor is connected to the input. When compared to Figure 3.7, the radix-2 burst architecture has reduced the amount of resources by only using one butterfly diagram. The component also has an advantage by heavily reducing the memory components from log2(N) to only 2.

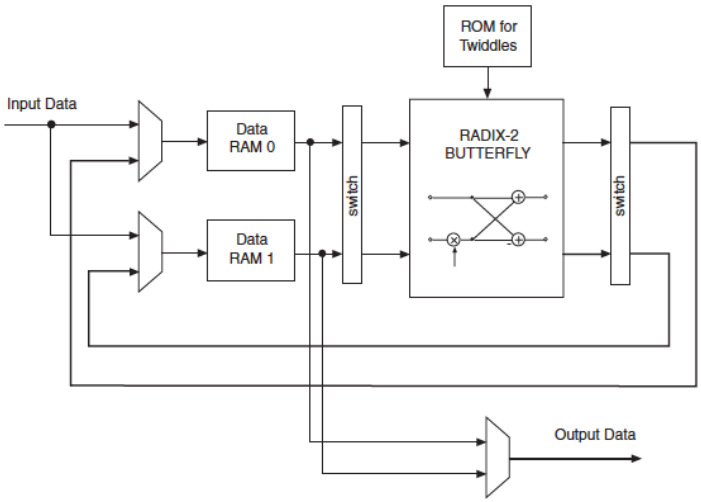


Figure 3.8: Architecture of radix-2 burst FFT. Source: [31]

One issue that must be taken into consideration is that since this architecture is designed to perform burst operations, the system cannot load constantly, as a compare to the pipelined streaming architecture that can load and unload constantly. The input data can only be loaded in bursts of N samples at a time, causing the system to have a wait-state that will reduce the loading capability of the architecture. The FFT component alerts any external system about this. Figure 3.9 show the timing diagram of the loading/processing/unloading process of the radix-2 burst architecture. When the ready signal goes low, i.e. the FFT has been loaded; the FFT goes in a wait state where it completes the processing and the unloading before loading a new frame.

Radix-2 Burst Lite The architecture of the radix-2 burst lite can be seen in Figure 3.10. As shown in the figure, the architecture has many strong

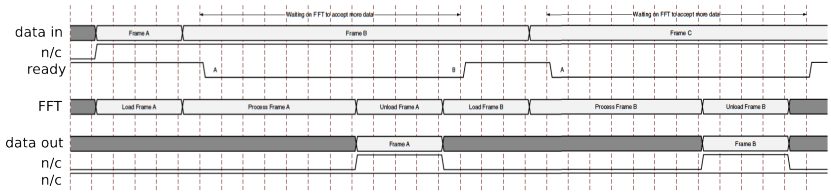


Figure 3.9: Timing diagram of wait state in the FFT radix-2 burst architecture. Source: [31]

similarities with the radix-2 burst architecture. However, there are some changes that have occurred. First, the most noticeable change is that all the switch components that could be observed in Figure 3.8 are removed. By doing so, the radix-2 burst lite architecture will provide the output data in reversed order instead of natural order. Second, two of three multiplexers are removed, and third, one of the RAM modules are removed. This modification causes the radix-2 burst lite architecture to have use the least resources of all the architectures but at a same time spend the most time to process the loaded input.

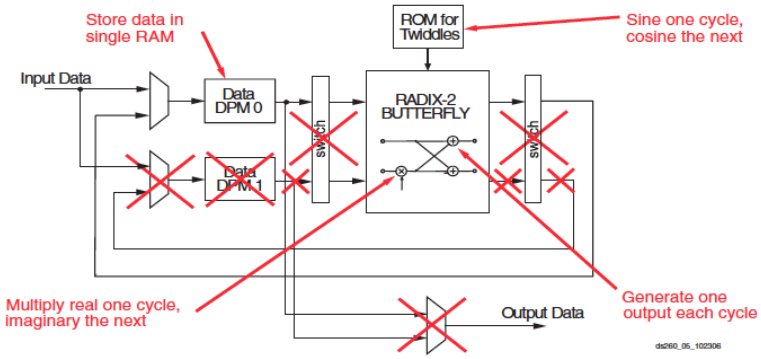


Figure 3.10: Architecture of radix-2 burst lite FFT. Source: [31]

Radix-4 Burst The fourth FFT architecture from Xilinx is the radix-4 burst architecture. It can be viewed in Figure 3.11 that the architecture has a more advanced processing structure, also known as the "dragonfly". Beside the larger processing component, the architecture follows the same architecture as the radix-2 burst design (see Figure 3.8). However, the architecture provides shorter processing time and reduced memory when compared to the radix-2 burst and radix-2 burst lite architecture.

Scaling Factor For the FFT architecture that has selected for this thesis, the scaling factor has only been implemented in the range compression component where the transformation length was 65,536 samples. This is because the dataset that were used to verify the functionality of the range compression component (Section 3.7.2), when converted from numeric values to binary values, does not cover a greater wordlength than maximum 10 bit complex data, ranging from -15

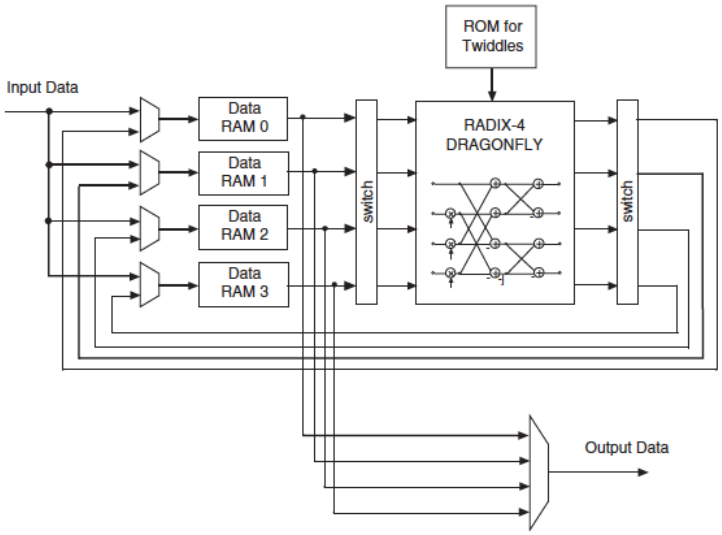


Figure 3.11: Architecture of radix-4 burst FFT. Source: [31]

to 15). A simulation was conducted in February 2012 and the samples from the final result ended up lying between -1 and 1. Therefore, the range compression component made for the simulation (where the transformation length is 2048 samples) was designed with an unscaled FFT/IFFT component.

The FFT transformations are configured using an own dedicated input channel. As mentioned earlier, this channel, the *configuration channel*, is mainly used to scale the data and select forward or inverse transformation. In [31], it is given an explanation of how to configure the FFT. It is with the configuration channel that the forward/inverse transformation is selected. The IFFT component is generated in the same manner as the FFT component.

There are two times a new configuration of the FFT can be applied, either to the very first frame after power on or after an idle period or to the next frame in a sequence of frames. For the Wavemill mission, it has been assumed that two separate FFT/IFFT components are used and therefore the configuration channels are loaded at the very first frame after power on and not modified beyond this point.

Matched Filter

The matched filter consists of a memory module that holds a replica of the transmitted pulse and a complex multiplier that multiplies the signal from the memory module with the signal from the FFT component. Figure 3.12 give a detailed view of how the matched filter is built.

Similar with the FFT, the complex multiplier were also first intended to be hard coded. The component showed promising results and was possible to implement within the FPGA without any remarkable errors. However, when being compared with a similar IP core, it was clear that the IP core was built far more advanced than the hard coded complex multiplier, in particular when

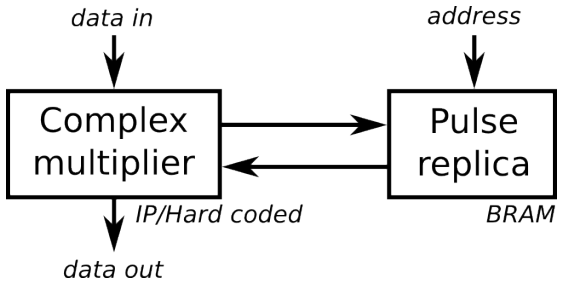


Figure 3.12: Detailed view of the matched filter

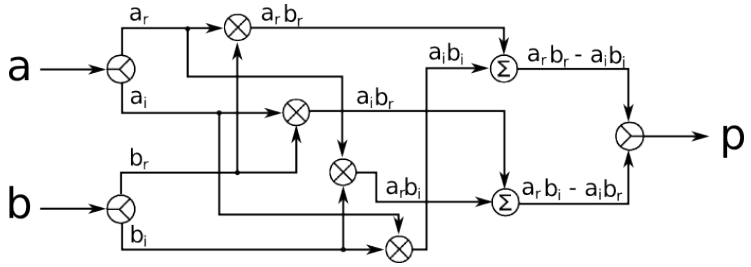


Figure 3.13: Detailed view of complex multiplier

it came to the usage of the BRAM blocks that was within the FPGA [30].

Two complex multipliers have been designed in this thesis. One hard coded version and one IP core version provided by the Core Generator [32]. The complex multiplier performs a mathematical operation where two complex input values are loaded and multiplied and a complex product is unloaded. The mathematical operation follows the calculation given in Equation 3.3.

$$
\begin{aligned}
a &= a_r + a_i \\
b &= b_r + b_i \\
ab &= a_r b_r + j(a_r b_i) + j(a_i b_r) - a_i b_i \\
p_r &= a_r b_r - a_i b_i \qquad (3.3) \\
p_i &= a_r b_i + a_i b_r \qquad (3.4)
\end{aligned}
$$

A visual presentation of Equation 3.3 can be seen in Figure 3.13. The figure assumes now that one of the inputs is from the memory component while the other is from the FFT component. Equation 3.3 shows an implementation where the two complex values a and b are multiplied directly together. This is the calculation that the hard coded complex multiplier has been based upon. The IP core exploits these calculations to generate a more simplified version to reduce the number of multiplications. Equation 3.5 shows the calculations the IP core is based upon.

$$
\begin{aligned}
p_r &= a_r(b_r + b_i) - b_i(a_r + a_i) \qquad (3.5) \\
p_i &= a_r(b_r + b_i) + b_r(a_r + a_i) \qquad (3.6)
\end{aligned}
$$

From Equation 3.5 it can be observed that the number of real multipliers is reduced from four to three, therefore reducing the resources needed to calculate the input. However, to see what impact the two different versions have on the overall system, both versions (hard coded complex multiplier and IP core complex multiplier) was synthesized.

The memory module is a BRAM block that has been coded as a Read-Only Memory (ROM) component. Two memory modules were designed for this thesis, one that contained 2048x32 bit complex samples and one that contained 65536x32 bit complex samples. In both cases, the 16 most significant bits were real while the 16 least significant bits were imaginary. The real and imaginary values followed a Q1.14 notation (`"http://en.wikipedia.org/wiki/Q_(numbering_format)"`). The pulse replica samples were generated in MATLAB and the script for generating the pulse replica can be found in Appendix A.4.

As with the FFT, the IFFT component was also first intended to be hard coded and designed after the principle given in [21]. However, due to the complications that occurred when hard coding the FFT component, an IP core IFFT were used in the thesis. The IFFT would have the same architecture as the FFT.

As described in the beginning of this chapter, four range compression components were designed in VHDL. Two of them were designed to verify the functionality of the design where one used a hard coded complex multiplier while the other used an IP core complex multiplier. The two components were designed for simulation purpose and both had a compression length of 2048 samples.

The other two range compression components were designed as according to the specifications given for the Wavemill mission (Section 3.4). This mostly involved expanding the BRAM blocks from 2048 complex samples to 65,536 complex samples. The complex multipliers used the same methodology as the simulation components. These two range compression components were designed for synthesis and synthesis only. This was because no raw SAR data from the Wavemill flight campaigns were available at the moment the range compression components were designed.

3.6.2 Azimuth Compression Component

The azimuth compression component was implemented as according to the proposed design given in Section 3.2. From Figure 3.1 it can be seen that the azimuth compression component had to perform the following:

- Complex multiplication (deramping)
- FFT
- Calculation of the phase (phase compensation)

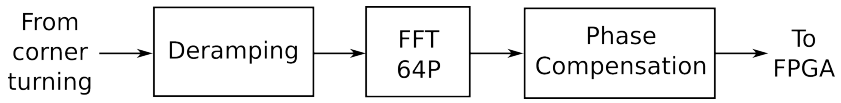


Figure 3.14: Azimuth compression component, architecture

Complex Multiplier, FFT and Phase Compensation

The complex multiplier and the FFT component follow the same description as given earlier in Section 3.6.1. The main difference is the length of the BRAM blocks and the transformation length of the FFT. As mentioned in Section 3.2, the azimuth samples will arrive at bursts of 50 samples per time. This is based upon the recommendation that was given in [18]. As mentioned in Section 3.4, an assumption is made that the burst length is 64 samples in azimuth. This was done to reduce the complexity of the system by avoid having a system to control the FFT component.

Since phase compensation requires a lot of processing power and resources, a simplified version had to be produced for this thesis. Following a recommendation by Astrium in[18], the phase compensation component was designed to be a BRAM block where the address would be the 6 most significant bits from the real and imaginary data to form a 12 bit word. The values in the BRAM block are generated in MATLAB by the use of its *atan2* function. In the MATLAB documentation, it states that this function calculates the inverse of the arctangent and the output lies in the range of $-\pi$ and π.

3.7 Simulation

Two simulations were conducted for this thesis: simulation of the four FFT architectures and simulation of the two range compression components. The goal for the first simulation was to find the least processing time of the architectures and the latter simulation was to verify the functionality and the design of the range compression component that was to be implemented for the Wavemill processor.

3.7.1 Overview Of The Simulation

The testbenches used to simulate the FFT architectures and the designed range compression components were based upon the FFT testbench that was generated by the Core Generator [31] when producing the FFT IP core. While the testbench used to simulate the FFT architectures was kept as it was, the testbench to simulate the range compression components were modified so that the data could be written to a file. MATLAB would then be used to visualize the data and a verification of the result could be made.

FFT

As briefly mentioned in this section, the testbench used to simulate the FFT architectures was kept as it was. The reason for this was that it was already made to verify the most important features to the FFT component (scaling, forward/inverse transformation). As stated in [31], "the testbench is a simple

VHDL testbench that exercises the core." The FFT architectures that were simulated had a transformation length of 65,536 samples.

The testbench was set to initialize the core, generate input values to be simulated, create a clock signal and drive the input signal to demonstrate the features of the core. The testbench generated an input of a complex sinusoid, and drive the configuration channel with different scaling and transformations (forward/inverse). The FFT components that were generated used an unscaled version and therefore the configuration channel would only drive a forward/inverse signal of the FFT. Since the FFT architectures were only made to be simulated to verify the statements about the processing time, as described in Section 3.6.1, any modifications of the testbench were not necessary.

Range Compression Component

The testbench used to verify the design and functionality of the range compression component had to be modified since some of the signals used in the FFT testbench was considered unnecessary for the range compression component. The most important modifications were to make it possible for the testbench to read from and write to a text file.

Figure 3.15 show the basic setup for the simulation. The test system reads values from a text file and each sample is transmitted to the range compression component. At the output, the test system writes each processed sample to a new text file that will be visualized in MATLAB.

Figure 3.15: Simulation range compression component, overview

Appendix B.1 shows the VHDL code written for the modified testbench used in the thesis. After initiation of the range compression component, the testbench opens the file that contains the necessary dataset to be processed (see Section 3.7.2 for description of the dataset) and set its mode in "read". The testbench also informs the range compression component that data is available. Then the testbench checks the file if there are more data available. If data can be extracted from the file, the testbench transfer the data from the file to the range compression component and repeat the operation until all data are extracted from the file. When this occurs, the process will close the file that contains the dataset and the entire process is ended. A state diagram for the read process can be seen in Figure 3.16.

Similar actions are conducted when the range compressed data is written to file. After initiation, the testbench opens a new file and sets the mode in "write". Note that any information that might be in the file will be overwritten when this is being performed. After creating the new file, the testbench checks if the range compression component has any processed data available on its output. If it has, the testbench will extract the data and write it to the file. If there are no data available, the testbench will wait for more processed data. A

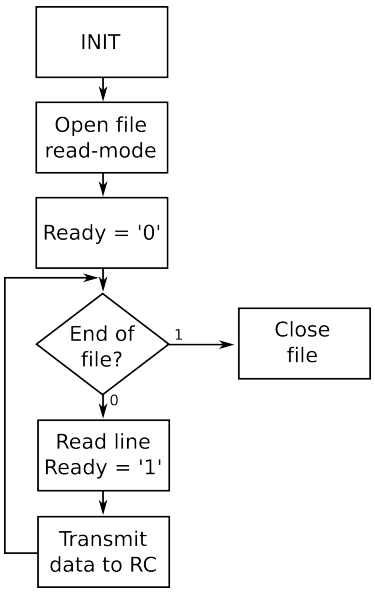


Figure 3.16: State diagram of read from file process

state diagram for the write-to-file process can be observed in Figure 3.17.

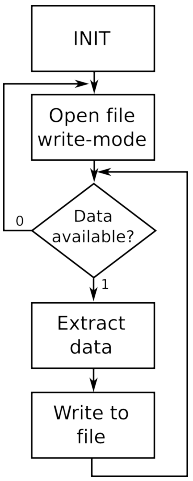


Figure 3.17: State diagram of write to file process

3.7.2 Dataset

The data that was transmitted to the range compression component (denoted "Data In" in Figure 3.15) were extracted from a dataset that was included in [19]. The dataset contained raw SAR data of the city of Vancouver, BC, Canada. Due to the size of the dataset, a small part was extracted to form a new dataset to be used in this thesis. The extracted dataset is the area of the Vancouver Ferry Terminal (Figure 3.18) and was acquired by the Canadian remote sensing

satellite Radarsat-1. The extracted dataset had an image size of 2048 (range) x 1536 (azimuth) complex samples.

Figure 3.18: Optical image of the Vancouver Ferry Terminal. Source: Google Maps

The SAR image was taken during night-time on June 16, 2002. The file was extracted from the dataset using pre-made MATLAB scripts from the website to [19] (`"http://www.artechhouse.com"`, search for "Ian Cumming"). The uncompressed dataset can be observed in Figure 3.19. A script was developed to convert the data from a 2 dimensional numeric complex value to a 1 dimensional complex binary text file. Each sample was 16 bits long whereas the 8 most significant bits were real and the 8 least significant bits were imaginary. The script can be found in Appendix A.1.

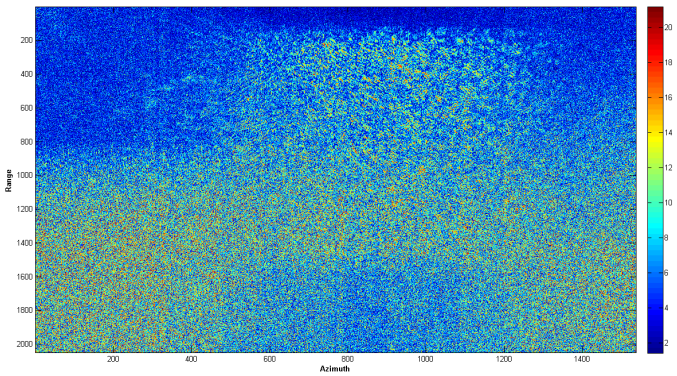


Figure 3.19: Raw SAR data of Vancouver Ferry Terminal

3.8 Synthesis

Seven syntheses have been conducted in this thesis. Four FFT architectures were synthesized to verify the statements about the resource demands in Section 3.6.1 and two range compression components were synthesized so it could be found out the impact of using a hard coded complex multiplier versus an IP core of the same component. Finally, the Wavemill real-time SAR processor was synthesized.

FFT

As described in Section 3.6.1, Xilinx provide four different FFT architectures. Each of the architectures has been configured in similar manners, i.e. no architecture is given an advantage over another. The Xilinx ISE Design Suite has been set up to provide a balanced relation between speed and resource optimization when synthesizing. The FFT architectures that were synthesized had a transformation length of 65,536 samples, i.e. the same as the FFT components used in the simulation.

Range Compression Component

The range compression component follows the same design as when being simulated. Only difference is that the compression length is now 65,536 instead of 2048. The set up for the component were also a balanced relation in regard to the optimization when synthesizing.

Azimuth Compression Component

The azimuth compression component follows the same design as described in Section 3.6.2.

SAR Processor

The SAR processor consisted of the range compression component and the azimuth compression component and was set up as according to Figure 3.20.

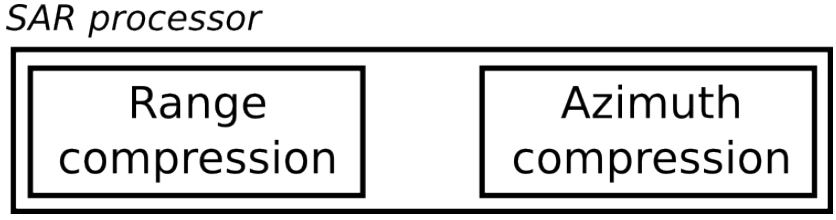


Figure 3.20: SAR processor synthesis setup

FPGA

The FPGA that the synthesis is targeted towards is the Xilinx Virtex-6 xc6vlx240-1ffg1759c. The FPGA is a part of the ESA-HSADC-EVAB Evaluation Board [33] that was available at the ESTEC facility.

3.9 Expected Results

This section will describe the expected results of the simulations and synthesis.

3.9.1 FFT Architectures

Four simulations and syntheses will be conducted on the four FFT architectures presented in Section 3.6.1. It is expected that the simulation and synthesis will verify the statement in [31] that the pipelined streaming FFT architecture will use the least time and require the most amount of resources from the FPGA with respect to the four architectures. It is also expected that the simulation and synthesis results will show that the radix-2 lite FFT architecture will spend the longest transformation time but require least amount of resources with respect to the four architectures. For the Wavemill mission, the most important information that can be extracted from the simulation is the transformation time while the synthesis will reveal the amount of resources that are necessary for the FFT component.

3.9.2 Range Compression

Figure 3.21 presents the range compressed dataset that is expected to be seen after the raw SAR data has been processed.

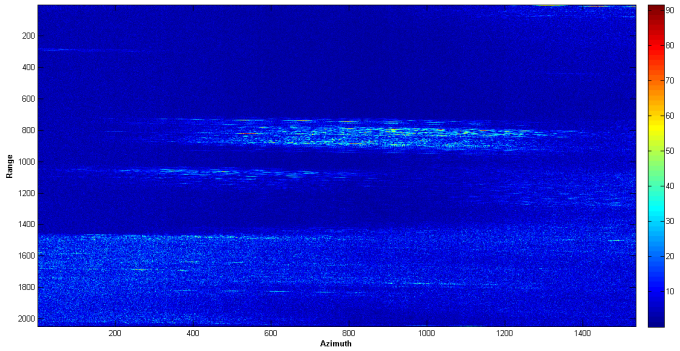


Figure 3.21: Expected range compressed raw data

From the figure, there can be observed some features that can be used as reference points. In this thesis, it was selected that the gathering of lines, located at 0 to 600 in the range direction (vertical line), would be used as reference points. A successful range compression would be if these features could be observed after the raw data has been range compressed by the SAR processor made in this thesis. This gathering was selected because of its dark areas that lies nearby that more than likely resemble waters and therefore the gathering is relative unique in the processed image.

Reference images were also produced for the FFT and the matched filter. These can be observed in Figure 3.22 and Figure 3.24 respectively. As can be

seen, there are difficulties finding any clear reference points in these two images, much because of the images appear to only contain interference, i.e. noise. However, two dark areas can be observed at the top left and top right of the reference images and will be used as references for the simulated results.

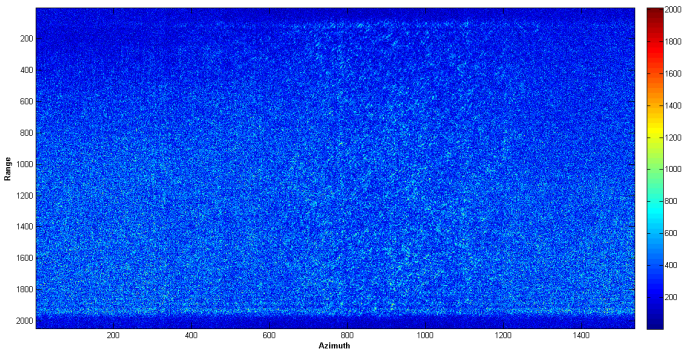


Figure 3.22: Expected FFT processed raw data

It should be noted that when processing the raw SAR data in MATLAB, it is normally assumed that the transformed data lies in the range of [-f_s/2, f_s/2) instead of [0,f_s). To achieve this, the MATLAB function *fftshift* is used to shift the data. In order to get correct data, as according to the design of the real-time SAR processor in this thesis, the *fftshift* function had to be dropped. This would make the simulation results (both FFT and matched filter) comparable with the reference images. Figure 3.23 show the FFT reference image without the *fftshift* function and will be the reference image for the simulation.

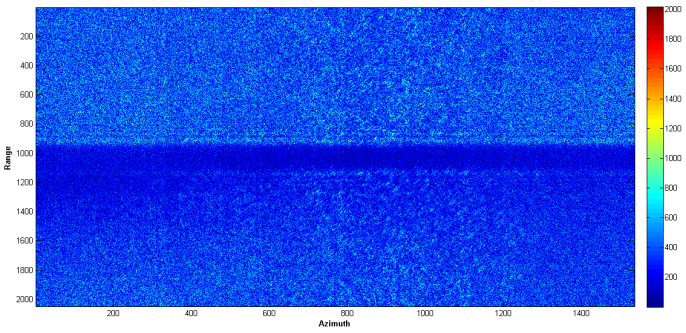


Figure 3.23: FFT operation of raw SAR data without the *fftshift* function

Figure 3.25 show the expected matched filter result without the *fftshift* function.

In this thesis, the simulation will be a success if the simulated range compression component could provide a result that corresponded with the expected results with regards of the pattern. However, a comparison between the two results would be performed to provide recommendations for further improvements of the system.

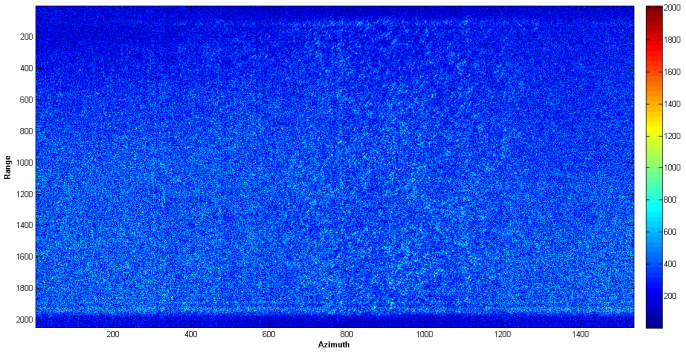


Figure 3.24: Expected filtered raw data

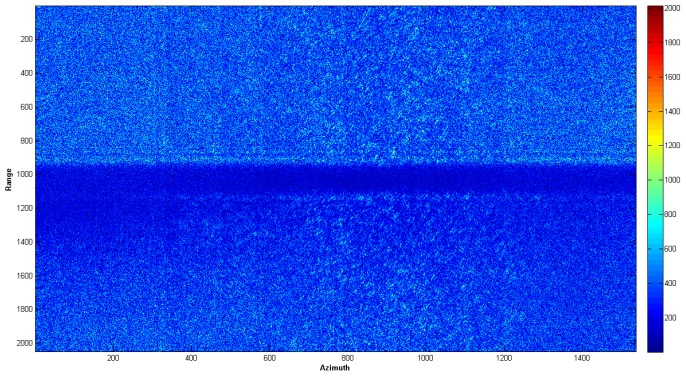


Figure 3.25: The figure shows the expected results after the raw SAR data has been filtered without the *fftshift* function.

Chapter 4

Simulation

The following chapter will present the simulation results for the four FFT architectures and the two range compression components. The background of the components were described in Chapter 2 and the implementation of the components and the setup for the simulation were described in Chapter 3. The results from the simulations will be discussed and commented in regards of the expectations that were described in Section 3.9.

The following simulations were conducted for this thesis:

- Simulation of four FFT architectures
  - Pipelined streaming FFT (65,536)
  - Radix-2 Burst FFT (65,536)
  - Radix-2 Burst Lite FFT (65,536)
  - Radix-4 Burst FFT (65,536)
- Simulation of range compression components
  - Complex multiplier was hard coded (2048)
  - Complex multiplier was IP core (2048)

The transformation length of each component is described in the parenthesis.

4.1 FFT architectures

The FFT architectures were simulated as according to the setup that was described in Section 3.7. The architectures used the same testbench so that it would be easier to compare the results. As stated in Section 3.9, it was expected that the pipelined architecture would spend the least processing time when transforming the input data.

The results after simulation are shown in Table 4.1. A total of 65,536 data samples[1] were transformed, as according to the design goals for the Wavemill

[1] generated by the testbench

processor. Note that a correct transformation was not taken into consideration at this simulation since this simulation would only be important in regard of the processing time for the different architectures.

Table 4.1 is divided into five columns. The first column show which architecture that has been simulated, the second column show the measured loading time the architecture has used, the third column show the processing time the architecture has spent, the fourth column show the measured unloading time and the fifth column show the summation of the three previous columns, load processing and out, to provide a total time the architecture spend on processing one frame, from end to end.

Architecture	**Load**	**Processing**	**Out**	**Total**
Pipe	6.553	6.569	6.553	19.677
R-2	6.553	52.455	6.553	65.562
R-2L	6.553	111.414	6.553	117.967
R-4	6.553	19.677	6.553	32.783

Table 4.1: Results FFT simulation. All values are in milliseconds (ms) and was measured with a clock period of 8 ns (125 MHz). The architectures were the following: Pipelined streaming (Pipe), Radix-2 Burst (R-2), Radix-2 Burst Lite (R-2L) and Radix-4 Burst (R-4).

From the table, we can observe that the simulated results verify the expected results that were mentioned in Section 3.9. We see that the pipelined streaming FFT architecture spend the least transformation time with respect to the architectures while the Radix-2 Burst Lite FFT architecture spend the most time.

4.1.1 Range compression

Five results are presented in this section: the result after the FFT, the result after the matched filter with hard coded and IP core complex multiplier and the result after the range compression component with hard coded and IP core complex multiplier. The raw SAR data that was used for simulation were described in Section 3.7.2.

FFT

The result from the FFT component is the same regardless if the range compression component contains a hard coded or IP core complex multiplier. This is because the transformation occurs before the matched filter component.

Figure 4.1 show the transformed signal after the FFT operation. As can be seen, there are clear pattern similarities between the expected result (Figure 3.23) and the simulated result.

Figure 4.2 show a comparison between the simulated FFT image and the adjusted MATLAB generated reference signal. The figure shows that there is a strong similarity between the expected result and the simulated result. The absolute average value between the expected result and the simulated results[2]

[2]ABS(simulated result) - ABS(expected result)

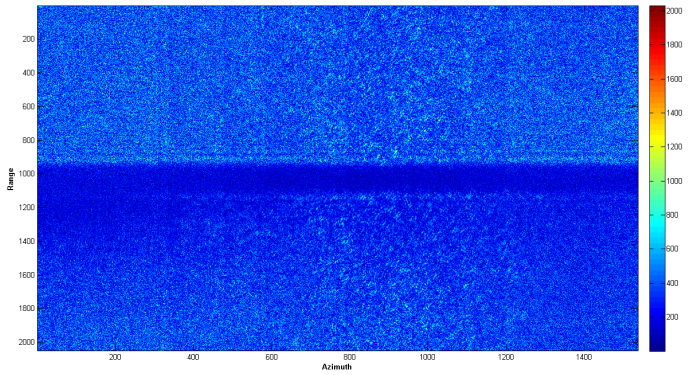


Figure 4.1: Simulated FFT of raw SAR data. The figure shows the absolute value of the complex samples after being transformed.

were calculated to be laying around 1, a value that can be considered a very good value since the results has a range between 0 and 2000 (see colorbar to the right of the figure).

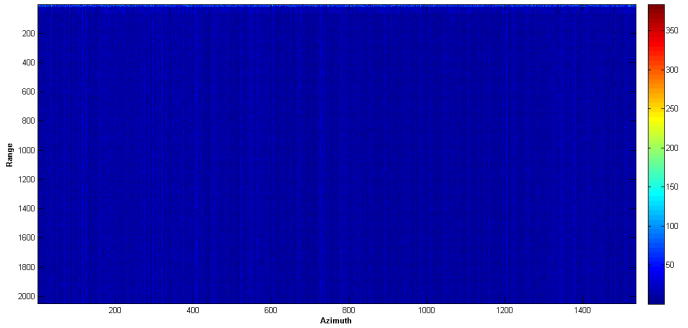


Figure 4.2: Difference between MATLAB and VHDL FFT

As mentioned in Section 3.9, the simulation would be assumed to be a success if the patterns were similar. As has been shown, the patterns from the simulated FFT and the expected result correspond and the result can be considered satisfactory.

4.1.2 Matched filter with hard coded complex multiplier

Figure 4.3 show the result after the raw SAR data has been filtered with the pulse replica. The filter used the hard coded complex multiplier that was described in Chapter 3. As can bee seen in the figure, there are strong similarities with the reference signal (Figure 3.25).

Figure 4.4 show the difference between the expected matched filter result and the simulation result. The figure shows that the difference between the simulation and the expected result has a larger variation than what the difference

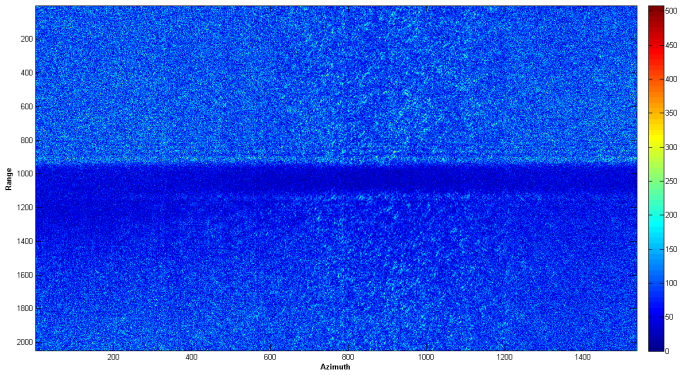


Figure 4.3: Simulated matched filter processed raw data. The data shown is the absolute value of the complex filtered sample points. The complex multiplier used in this simulation was hard coded.

between the simulated FFT and the expected FFT result. The absolute average value of the difference between the matched filters were calculated to be around 251.

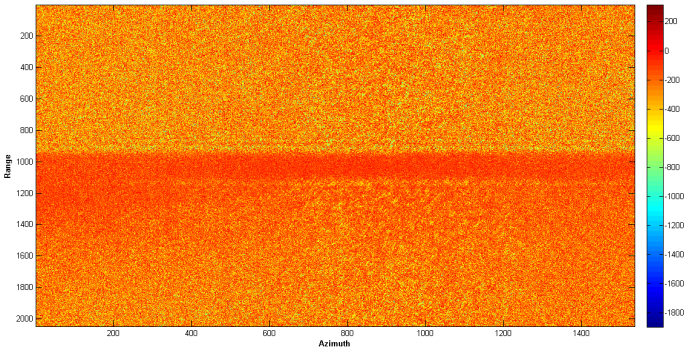


Figure 4.4: Difference between expected result and simulation MF, hard coded complex multiplier

A modification to the simulated matched filter result, i.e. scaled up with a factor of 2^2 caused the data to be more precise to the expected result. The difference can be seen in Figure 4.5 and were calculated to have an average value laying near 1, a decrease by 250 when compared to the previous unscaled difference.

The expected patterns are recognizable in the simulated result, and despite the high difference in the amplitude values, the result is regarded as satisfactory with respect to the simulation goals.

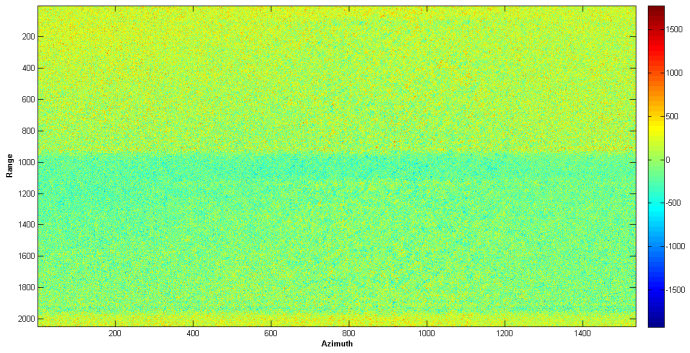


Figure 4.5: Difference between expected result and simulation matched filter, hard coded complex multiplier. The result from the simulation was scaled with a factor of 2^2.

4.1.3 After matched filter with IP core complex multiplier

Figure 4.6 show the result where the data has been filtered; only this time the filter included the IP core version of the complex multiplier. As can be seen in the figure, the two dark spots that were defined as reference points in Section 3.9 appear in the result.

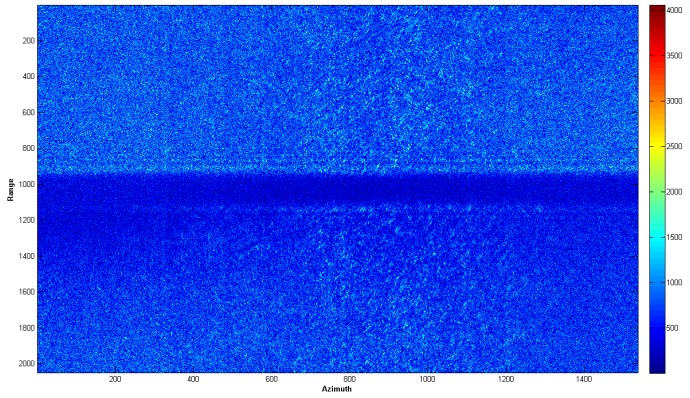


Figure 4.6: Matched filter processed raw data where the complex multiplier was an IP core.

Figure 4.7 show the difference between the expected result and the IP core processed matched filter. The absolute average difference was calculated to be laying around 337. However, when scaling the simulated results, i.e. the result was multiplied with a factor of $1/2^2$, the absolute average difference was calculated to be 2, a more acceptable value for the final results. The result can be seen in Figure 4.8.

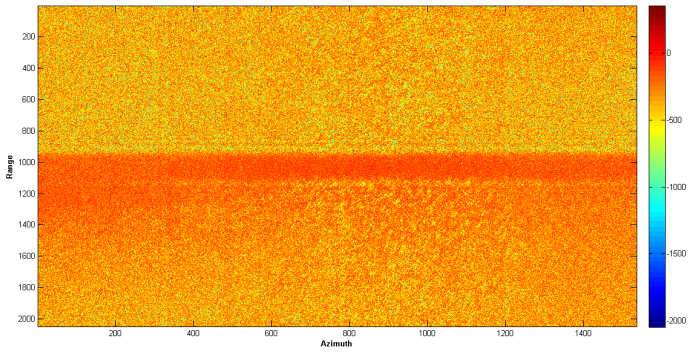


Figure 4.7: Difference between expected result and simulation matched filter, IP core complex multiplier

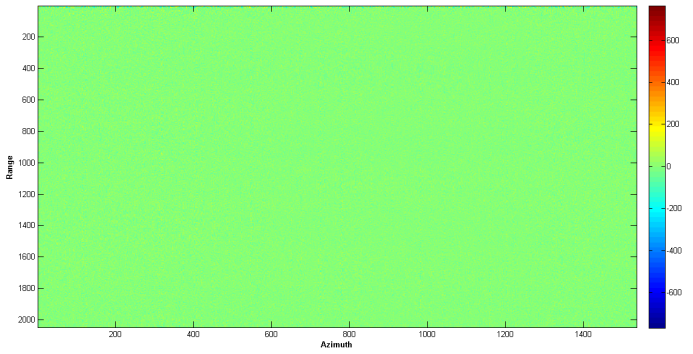


Figure 4.8: Difference between expected result and simulation matched filter after being scaled, IP core complex multiplier

4.1.4 Range Compressed with hard coded complex multiplier

Figure 4.9 show the result where the raw SAR data has been range compressed with the hard coded complex multiplier. When comparing the pattern in the simulation result with the expected result, as presented in Figure 3.21, it appears that the designed range compression component has strong similarities with the expected result. However, by having a look at the colorbar on the right side of the image, it shows that the range is very large when compared to the range of expected result (Figure 3.21).

Figure 4.10 show the difference between the expected result and the simulation result with hard coded complex multiplier. It becomes clear that there is a large difference between the expected result and the simulated results. A calculation show that the absolute average difference between the two results are as high as 3539,9. When scaling the simulated data with a factor of 2^9 we get a more acceptable result, as shown in Figure 4.11. The absolute average difference was now calculated to be 0.4316, a value more acceptable to what

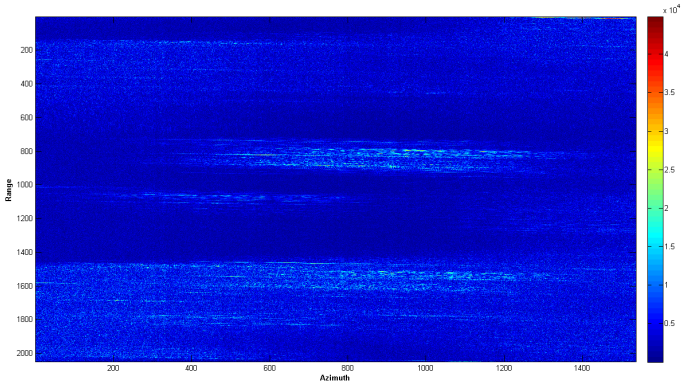


Figure 4.9: Range compressed data with hard coded complex multiplier

should be expected (as close to 0 as possible).

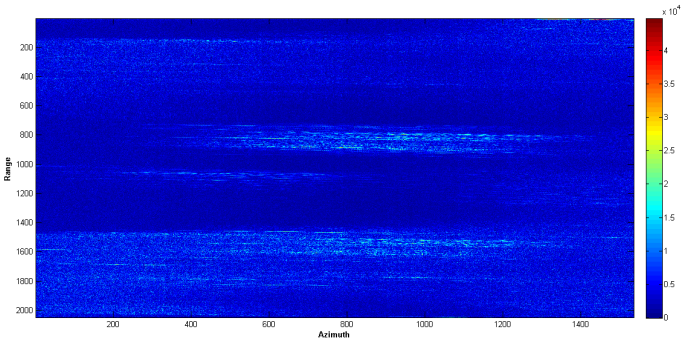


Figure 4.10: Difference between expected result and simulation RC, hard coded complex multiplier

One important point that must be taken into consideration is the pattern similarities, because despite the fact that the ranges (shown in the different colorbars) are different from the simulated results and the expected results, the patterns have proven to be similar. From the consideration of satisfaction, the patterns will weight most in regard of the conclusion of satisfaction of the results. Therefore the results are satisfactory despite inequalities in the amplitude level.

4.1.5 Range Compressed with IP core complex multiplier

Figure 4.12 show the range compressed dataset where an IP core of the complex multiplier has been used.

Figure 4.13 show the difference between the expected result and the simulation result. The absolute average difference was calculated to be 2837.1, a large number that indicates a high variation. As with the matched filter, a scaling was performed on the simulated results, dividing the values with a factor of 2^{12}.

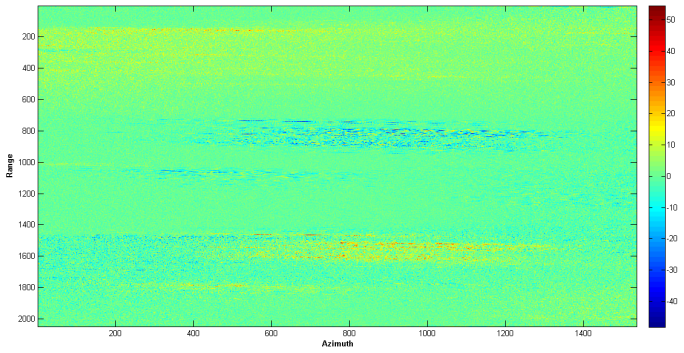


Figure 4.11: Difference between expected result and simulation RC, hard coded complex multiplier

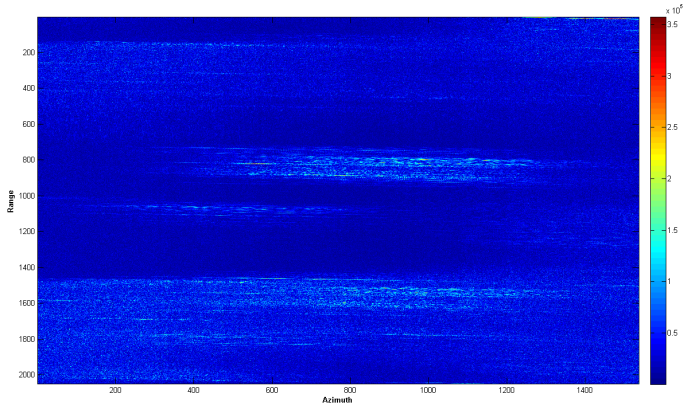


Figure 4.12: Range compressed data with IP core complex multiplier

The difference was then calculated to be 0.4331. The difference can be seen in Figure 4.14.

4.2 Discussion

As has been commented in this chapter, the amplitude level of the simulated results is much higher than what was expected, then particular after the matched filter component and the IFFT component. As stated in Section 4.1.2, only the most significant bits were extracted from the complex multiplier. This can also be the reason why the scaling factors had their respective values. A modification of this, where the least significant bits were extracted (from bit number 15 and higher), would be preferable for a future simulation.

Simulations also showed that the IP core complex multiplier had a higher absolute difference value than the hard coded complex multiplier, 2 and 1 respectively. A reason for this is how the calculations are performed. As mentioned

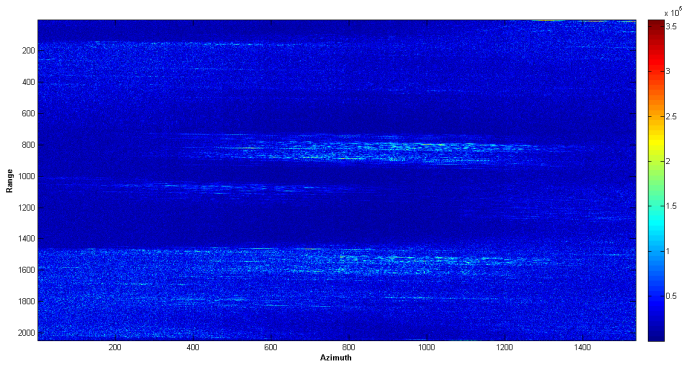


Figure 4.13: Difference between expected result and simulation RC, IP core complex multiplier

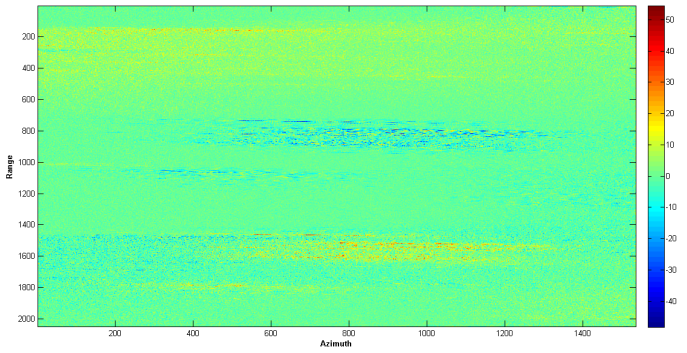


Figure 4.14: Difference between expected result and simulation RC, IP core complex multiplier

in Section 3.6.1, the IP core use a modified algorithm to perform the complex multiplication and this might be the cause of the difference. However, a future simulation would be preferable to verify this statement.

The simulations can be concluded to be satisfactory. This is because of the pattern that can be observed in the expected results can also be observed in the simulated results. However, improvements are needed to reduce the amplitude level of the result. This will be mentioned in Chapter 6.

Chapter 5

Synthesis

This chapter will present the results from the synthesis of the different FFT architectures, the range and azimuth compression components and the Wavemill real-time SAR processor.

5.1 FFT

Table 5.1 show the results after synthesizing the four different FFT designs. From the table, we see that the pipelined streaming FFT design requires much more resources than the other three FFT designs.

Table 5.1 shows that the assumptions made in Chapter 3 were correct; the pipelined streaming architecture requires most resources when compared to the other FFT architectures. The table shows that the architectures with the highest clock frequency (Radix-2 Burst Lite and Radix-2 Burst) are the same architectures that requires the least amount of resources.

5.2 Synthesis of scaled/unscaled FFT component

As mentioned in Chapter 3, after finding the optimal FFT architecture that had the least processing time without a large demand for resources, finding the optimal FFT component was the next step. Two synthesis were conducted on two balanced (optimization) pipelined streaming FFT architectures, one with

FFT	**Freq**	**Reg**	**LUTs**	**Mem**	**BRAM18**	**BRAM36**	**DSP48**
Pipe	350	2.29	3.45	2.02	25.36	6.49	2.73
R-2	416	0.42	0.71	0.18	11.53	4.32	0.39
R-2L	416	0.34	0.43	0.17	0	17.3	0.26
R-4	411	0.82	1.31	0.38	13.9	5.76	1.17

Table 5.1: Synthesis FFT designs. The shortened parameters are frequency (Freq), registers (Reg) and memory (Mem). All values except for the frequency are in per cent. The frequency value is in MHz. The FFT components are as follows (from top to bottom): pipelined streaming (Pipe), radix-2 (R-2), radix-2 lite (R-2L) and radix-4 (R-4).

Parameter	Scaled	Unscaled
Max clock frequency (MHz)	400.962	400.962
Registers	1.99%	1.45%
LUTs	3.21%	2.30%
Memory	2.63%	1.81%
BRAM18	28.2%	12.1%
BRAM36	3.6%	4.5%
DSP48	2.73%	2.73%

Table 5.2: Synthesis results, optimized 65K FFT

Parameter	Hard coded	IP core
Max clock frequency (MHz)	134.673	242.189
Slice registers	4%	5%
Slice LUTs	8%	8%
-Memory	3%	3%
Occupied Slices	12%	12%
LUT FF pairs	14,703	14,727
RAMB36E1s	16%	16%
RAMB18E1s	25%	25%
DSP48E1s	1%	1%

Table 5.3: Range compression results, synthesis

a scaled FFT component and another one with an unscaled FFT component. The synthesis results can be observed in Table 5.2

In regard of [18] and the Wavemill mission, the pipelined streaming FFT architecture was selected to be implemented in the range compression component. This was because the sampled data is arriving constantly to the system, requiring a system that must process directly without any wait states. In regard of the four FFT architectures, the pipelined streaming architecture is the only one that can do that.

5.3 Synthesis of range compression component

The range compression component that was synthesized was the Wavemill real-time SAR processor that was described in Section 3.6.1. The range compression component had a length of 65,536 points, as according to the design goals mentioned in Section 3.4. Two syntheses were conducted, one with hard coded complex multiplier and one with IP core complex multiplier.

5.4 Synthesis of azimuth compression

The synthesis of the azimuth compression component were also conducted in a similar manner as the range compression component mentioned earlier, with both a hard coded complex multiplier and a IP core of the same component. This was to find out the measurements and to see if similar information could be ex-

Parameter	Hard coded	IP core
Max clock frequency (MHz)	526.445	543.478
Slice registers	1%	1%
Slice LUTs	1%	1%
-Memory	1%	1%
Occupied Slices	4%	4%
LUT FF pairs used	1,023	980
RAMB36E1s	1%	1%
RAMB18E1s	1%	1%
DSP48E1s	1%	1%

Table 5.4: Azimuth compression results, synthesis

Parameter	All IP	RC: HC	All HC
Max clock frequency (MHz)	242.189	134.673	134.673
Slice registers	5%	5%	5%
Slice LUTs	9%	9%	9%
-Memory	4%	4%	4%
Occupied Slices	13%	13%	13%
LUT FF pairs used	15,763	15,799	15,661
RAMB36E1s	17%	17%	17%
RAMB18E1s	25%	25%	25%
DSP48E1s	1%	1%	1%

Table 5.5: Wavemill processor results, synthesis

tracted. Table 5.4 show the results after the synthesis of these two components. The azimuth compression component was based upon the design described in Section 3.6.2.

5.5 Wavemill Real-Time Processor

The last synthesis that was conducted was with different versions of the Wavemill SAR processor. The synthesis was as following:

- All components are IP cores
- The range compression component use a hard coded multiplier (HC)
- All compression components use hard coded multipliers

Table 5.5 show the results after the synthesis.

5.6 Discussion

As seen in Table 5.3, it is clear that the IP core component use a little more resources than the hard coded complex multiplier but it gains a maximum clock frequency that is over 100MHz faster. A reason for this might be that the architecture exploits the BRAM blocks and available resources more efficient.

Another reason might be, as mentioned in Chapter 3, the IP core only use three real multipliers and therefore reduces the processing time and digital logic. It can also be observed from Table 5.5 that the hard coded complex multipliers affect the overall processing of the SAR processor with respect of the clock frequency.

Chapter 6

Discussion

This chapter provide a discussion regarding the overall SAR processor that has been implemented in the thesis. It also presents recommendations and improvements for future work on the SAR processor. A final conclusion for the thesis and the work that has been put forth will finish the thesis.

6.1 Results

The results from the synthesis proved that the system proposes SAR processor could be implemented in an FPGA. However, it became clear from the simulation results that the SAR processor is far from complete. As described in Section 4.1.1, the range of the amplitude level is higher than expected in regard of the simulated results when compared with the reference signal. Since the patterns from the simulated results show similarities with the expected results, it can be assumed that this occurs because of the extracted value from the different components, as described in Section 4.2. As shown in Section 4.1.1, a scaling of the simulated data causes the difference to decrease while the similarities remain the same.

6.2 Proposed SAR processor

Synthesis showed that the proposed SAR processor could be implemented in an FPGA. However, as described in Section 3.4, assumptions had to be made in regard of the samples in both range direction and azimuth direction. An implementation of the SAR processor as according to the data found in [18] would require an implementation of a control system to control the system of when to process and when not to process. This would be recommendable for a future implementation and simulation.

6.3 Future Work and improvements

- Range Compression Component
  - Despite showing promising results, there are possibilities to improve the systems at some of its aspects. A more in-depth simulation of the

different components is recommendable, then particular with a control system involved. It is also recommended that the range compression component is improved to make sure that the produced results correspond better with the reference images.

- Azimuth Compression Component
  - An improvement of the azimuth component should be achievable. The component should also be simulated to verify its function. Since the azimuth component relies on the SAR algorithm used, improvement of the algorithm, or modifications, should be implemented.

6.4 Conclusion

The main goal for this thesis has been to develop a real-time SAR processor for ESAs Wavemill mission. The thesis has described the theory, design and implementation of a modified SPECAN SAR processor that was recommended to be used for Wavemill. A range and azimuth compression component were designed and implemented for a Xilinx Virtex-6 FPGA.

Simulations and synthesis were conducted on the necessary components and showed promising results for a future real-time SAR processor. The simulations were conducted on the range compression components since the azimuth compression component was still under development during the thesis. Several simulations were conducted with hard coded components and IP core components. The simulations showed that the results corresponded with the expected results but further improvements and developments are needed before the system is implemented in a physical SAR system.

Synthesis showed that IP core components would be preferable to implement due to their exploit of available resources which gained a higher clock frequency than the hard coded components. Synthesis showed that the overall system would work under at least 240 MHz when using IP core components and 134 MHz when using hard coded components.

The thesis concludes that the designed SAR processor, by synthesis results, can be implemented in one single FPGA and can be further used for ESAs Wavemill mission. It is desirable that the work that has been put forth in this thesis will be used for further development of the SAR processor.

Bibliography

[1] J. C. Curlander and R. N. McDonough, *Synthetic Aperture Radar Systems and Signal Processing*. John Wiley and Sons, Inc., 1991.

[2] G. Franceschetti and R. Lanari, *Synthetic Aperture Radar Processing*. CRC Press, first ed., 1999.

[3] C. A. Wiley, "Pulse doppler radar methods and apparatus," 1954. US Patent no. 3,196,436.

[4] S. Holm. Personal communication, 2011.

[5] C. Buck, M. Aguirre, C. Donion, D. Petrolati, and S. D'Addio, "Steps towards the preparation of a Wavemill mission," in *Geoscience and Remote Sensing Symposium (IGARSS), 2011 IEEE International*, pp. 3959 – 3962, july 2011.

[6] M. Suess, C. Schaefer, and R. Zahn, "Discussion of the introduction of on-board sar data processing to spaceborne sar instruments," in *Geoscience and Remote Sensing Symposium, 2000. Proceedings. IGARSS 2000. IEEE 2000 International*, vol. 5, pp. 2331 – 2333 vol.5, 2000.

[7] N. Oceanographic and A. Administration, "How are currents measured?." `http://oceanservice.noaa.gov/education/kits/currents/07measure1.html`, March 2008.

[8] S.-R. Lee, "Overview of kompsat-5 program, mission, and system," in *Geoscience and Remote Sensing Symposium (IGARSS), 2010 IEEE International*, pp. 797 –800, july 2010.

[9] E. S. Agency. "`http://envisat.esa.int/handbooks/asar/CNTR4.htm#eph.asar.faq:smallest`", n.d.

[10] E. Rodriguez, B. Pollard, and M. J., "Wide-swath ocean altimetry using radar interferometry," in *IEEE Transactions on Geoscience and Remote Sensing Symposium, IGARSS'99*, jul 1999.

[11] S. B. S.L., *Final Report*. Starlab Barcelona S.L., Dec. 2008.

[12] B. P. R. C. at The Ohio State University. "`http://bprc.osu.edu/water/mission_overview.php`", n.d.

[13] C. Buck, *Wavemill Concept, ESA Technical Note, TEC-ETP/2004.121/CBu*. European Space Agency, 2004.

[14] J. Fontanet, B. Chapron, L. Romero, and J. Marquez, *Wavemill: WP100 - D1 Determination of Scientific Requirements*. Ifremer and Starlab, 2008.

[15] R. Barstow, *Envisat-1 Products Specifications*. European Space Agency and Macdonald Dettwiler, 2007.

[16] C. T. C. I. for Marine and A. Studies, "Ocean surface currents." "`http://oceancurrents.rsmas.miami.edu`", n.d.

[17] A. R. Robinson and K. H. e. Brink, *The Global Coastal Ocean: Multiscale Interdisciplinary Processes*. Harvard University Press, 2005.

[18] A. Limited, *Wavemill : Instrument Design Document*. EADS Astrium, 2008.

[19] I. G. Cumming and F. H. Wong, *Digital Processing of Synthetic Aperture Radar Data*. Artech House Inc., first ed., 2005.

[20] Wikipedia, "Electromagnetic spectrum." `http://en.wikipedia.org/wiki/Electromagnetic_spectrum`.

[21] A. Ambardar, *Digital Signal Processing - A Modern Introduction*. Thomson Ltd., international student ed., 2007.

[22] A. Thompson, J. Curlander, N. McLagan, T. Feather, M. D'Iorio, and J. Lam, "Scansar processing using the fastscan system," in *Geoscience and Remote Sensing Symposium Proceedings, 1994. IGARSS '94. 1994 IEEE International*, vol. 2, pp. 1187 – 1189, vol.2, aug. 1994.

[23] X. Yizhuang and L. Teng, "Implementation of spaceborne SAR imaging processor based on FPGA," in *Signal Processing, 2008. ICSP 2008. 9th International Conference on*, pp. 2318 –2321, oct. 2008.

[24] C. Le, S. Chan, F. Cheng, W. Fang, M. Fischman, S. Hensley, R. Johnson, M. Jourdan, M. Marina, B. Parham, F. Rogez, P. Rosen, B. Shah, and S. Taft, "Onboard FPGA-based SAR processing for future spaceborne systems," in *Radar Conference, 2004. Proceedings of the IEEE*, pp. 15 – 20, april 2004.

[25] C. Y. Chang, W. C. Fang, and J. C. Curlander, "Data compression for eos on-board SAR processor," in *1989 International Geoscience and Remote Sensing Symposium, 1989. IGARSS '89. 12th Canadian Symposium on Remote Sensing.*, vol. 3, pp. 1723 – 1728, July 1989.

[26] N. Nolte, C. Simon-Klar, S. Langemeyer, and P. Pirsch, "Architecture of a flexible on-board real-time SAR-processor," in *Proceedings. 2005 IEEE International Geoscience and Remote Sensing Symposium, 2005. IGARSS'05.*, vol. 3, pp. 1746 – 1749, July 2005.

[27] D. Microwaves and R. Institute, "E-sar - the airborne sar system of dlr." `http://www.dlr.de/hr/en/desktopdefault.aspx/tabid-2326/3776_read-5679/`, 2012.

[28] D. Microwaves and R. Institute, "E-sar - the airborne sar system of dlr." `http://www.dlr.de/hr/en/desktopdefault.aspx/tabid-2326/3776_read-5691/`, 2012.

[29] I. Kuon and J. Rose, "Measuring the gap between FPGAs and ASICs," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 26, pp. 203 – 215, Feb. 2007.

[30] Xilinx, *IP Processor Block RAM (BRAM) Block (v1.00a)*, March 2011.

[31] Xilinx, *LogiCORE IP - Fast Fourier Transform v8.0*, March 2011.

[32] X. Inc., *LogiCORE IP - Complex Multiplier v4.0*. Xilinx Inc., Mar. 2011.

[33] P. E. D. GmbH, *ESA-HSADC-EVAB Evaluation Board - User Manual*, Apr. 2011.

Appendix A

M-files

A.1 Write_to_file.m

The following MATLAB script takes the data from the MAT-file *Ferry_terminal* and converts it from 2D complex integer values to 1D vector complex 16 bit binary vector.

```
% Raw data from RADARSAT-1
load Ferry_terminal.mat

% Separate data in real and imaginary parts
% Section 1
data_re = real(data);
data_im = imag(data);

% Extract sign for each value
sign_re = sign(data_re);
sign_im = sign(data_im);

% Convert to absolute values
data_re = abs(data_re);
data_im = abs(data_im);

% Generate an empty vector
data_re_new = zeros(length(data.'), length(data));

% Section 2
for i=1:length(data)
    for j=1:length(data(:,i))
        if (sign_re(j,i) == -1)
            val = data_re(j,i);
            tmp = bitcmp(uint8(val),8)+1;
        else
            tmp = data_re(j,i);
        end

        data_re_new(j,i) = tmp;
    end
end

for i=1:length(data)
    for j=1:length(data(:,i))
```

```
        if (sign_im(j,i) == -1)
            val = data_im(j,i);
            tmp = bitcmp(uint8(val),8)+1;
        else
            tmp = data_im(j,i);
        end

        data_im_new(j,i) = tmp;
    end
end

% File parameters
filename = 'Data_Radarsat1.txt';
fid = fopen(filename,'w');

data = data.';
data_re_new = data_re_new.';
data_im_new = data_im_new.';

% Section 3
for i=1:length(data(i,:)) % For every row
    disp(['Row number ' i ' of 1536']);
    for j=1:length(data) % For every column
        fprintf(fid,dec2bin(data_re_new(j,i),8));
        fprintf(fid,dec2bin(data_im_new(j,i),8));
        fprintf(fid,'\n');
    end
end

fclose(fid);
```

A.2 Phase_memory.m

The following MATLAB script generates the different phase compensation variables that is transmitted to the output of the SAR processor.

```
filename_bin = 'Phase_reference_bin.txt';
fid_bin = fopen(filename_bin,'w');

max_length_I = 2^6;
max_length_Q = 2^6;

memory = zeros(1,max_length_I*max_length_Q);

for I=1:max_length_I
    for Q=1:max_length_Q
        memory(:,(I-1)*2^6+Q) = atan2((Q-1),(I-1))*2^14;
    end
end

% Write to file
for i=1:length(memory)
    fprintf(fid_bin,'phase_out(%i) <= ', (i-1));
    fprintf(fid_bin, dec2bin(floor(memory(1,i)),16));
    fprintf(fid_bin,';\n');
end

fclose(fid_bin);
```

A.3 Pulse_replica_generator.m

This file generates 65K 16 bit complex values to represent a sine tone. The values are "dummy" values, only to be used when synthesized so that an estimate of resources can be calculated.

```
maxrg2 = 65536;
Fr     = 1;
Kr     = 1;

time    = 0:maxrg2-1;
arg     = -1i*2*pi*time/maxrg2;

Sine_signal = exp(arg);

filename = 'Pulse_replica.txt';
fid      = fopen(filename,'w');

for i=1:length(Sine_signal)
    sign_real = sign(real(Sine_signal(i)));
    sign_imag = sign(imag(Sine_signal(i)));

    fprintf(fid,['  replica(' num2str(i-1) ') <= "']);

    if (sign_real == -1)
        newMF = bitcmp(abs(floor(real(Sine_signal(i))*128)),8)+1;
        fprintf(fid,'%s',dec2bin(newMF,8));
        % fprintf(fid,' ');
    else
        % fprintf(fid,'%s',['  replica(' num2str(i-1) ') <= "' dec2bin(real(MF(i)),16) '";']);
        fprintf(fid,'%s',dec2bin(real(Sine_signal(i))*128,8));
        % fprintf(fid,' ');
    end

    if (sign_imag == -1)
        newMF = bitcmp(abs(floor(imag(Sine_signal(i))*128)),8)+1;
        % fprintf(fid,'%s',['  replica(' num2str(i-1) ') <= "' dec2bin(newMF,16) '";']);
        fprintf(fid,'%s',dec2bin(newMF,8));
        %fprintf(fid,'\n');
    else
        % fprintf(fid,'%s',['  replica(' num2str(i-1) ') <= "' dec2bin(imag(MF(i)),16) '";']);
        fprintf(fid,'%s',dec2bin(imag(Sine_signal(i))*128,8));
        %fprintf(fid,'\n');
    end
    fprintf(fid,'";\n');
end

fclose(fid);
```

A.4 Pulse_replica_generator_MF.m

This file generates the pulse replica for the range compression component. The values that are generated have a wordlength of 32 bit where the 16 most significant bits are real and the 16 least significant bit are imaginary. Both the real and imaginary values follows the Q1.14 notation (signed word, 1 integer bit, 14 fractional bit).

```
% Parameters
Kr = 7.213500000000000e+11;
Fr = 32317000;
maxrg2 = 2048;

% Generate pulse replica
f_tau = Fr*(-maxrg2/2:maxrg2/2-1)/maxrg2; % Page 172
arg     = -1i*pi*f_tau.^2/Kr;

MF = exp(arg);

% Extract signs
sign_real = sign(real(MF));
sign_imag = sign(imag(MF));

% Generate a textfile where data will be stored
filename = 'Pulse_replica.txt';
fid      = fopen(filename,'w');

% Convert to Q1.14 and write to file
for i=1:length(MF)
    fprintf(fid,['  replica(' num2str(i-1) ') <= "']);

    if (sign_real(i) == -1)
        newMF = bitcmp(abs(floor(real(MF(i))*2^14)),16)+1;
        fprintf(fid,'%s',dec2bin(newMF,16));
    else
        fprintf(fid,'%s',dec2bin(floor(real(MF(i))*2^14),16));
    end

    if (sign_imag(i) == -1)
        newMF = bitcmp(abs(floor(imag(MF(i))*2^14)),16)+1;
        fprintf(fid,'%s',dec2bin(newMF,16));
    else
        fprintf(fid,'%s',dec2bin(floor(imag(MF(i))*2^14),16));
    end
    fprintf(fid,'";\n');
end

fclose(fid);
```

A.5 Range_compression.m

```
clear, home, format compact

close all

load CD_run_params
load Ferry_terminal_1.mat

datain = double(data);

Fr     = 32317000;
Kr     = 7.2135e+11

maxrg  = 2048;
maxaz  = 1536;
```

```
maxaz2 = 2048;
maxrg2 = 2048;

data = zeros(maxrg,maxaz) + 1i*zeros(maxrg,maxaz);
data(1:maxaz,1:maxrg) = datain;

dataout = fftshift(fft(data,[],2),2);

f_tau = Fr*(-maxrg2/2:maxrg2/2-1)/maxrg2;
arg     = -1i*pi*f_tau.^2/Kr;

MF = exp(arg);

for i=1:maxaz
    dataout(i,:) = dataout(i,:) .* MF;
end;

dataout = ifft(dataout,[],2);
```

Appendix B

VHDL-files

B.1 Testbench Range Compression

```
-------------------------------------------------------------------------
-- Generate clock
-------------------------------------------------------------------------

clock_gen : process
begin
  aclk <= '0';
  wait for CLOCK_PERIOD;
  loop
    aclk <= '0';
    wait for CLOCK_PERIOD/2;
    aclk <= '1';
    wait for CLOCK_PERIOD/2;
  end loop;
end process clock_gen;

-------------------------------------------------------------------------
-- Function that reads the file 'Data_Radarsat1.txt' and extracts
-- 1 and 1 sample at a time. If FFT informs the system that it is busy,
-- the system will hold and wait to signal goes low.
-------------------------------------------------------------------------

-------------------------------------------------------------------------
-- Extract data from file 'Data_Radarsat1.txt'
-------------------------------------------------------------------------

process
  begin
    reset <= '1','0' after 100 ns;
    wait;
end process;

data_stimuli : process
file filein          : text;
  variable input_line  : LINE;
  variable input       : std_logic_vector(15 downto 0);
begin

  Datain_tvalid <= '0';
```

```
    wait until rising_edge(aclk);
  file_open(filein, "C:\..\Data_Radarsat1.txt", read_mode);

  while(not(endfile(filein))) loop
    Datain_tvalid <= '1';
    wait until rising_edge(aclk);

    readline(filein,input_line);
    read(input_line,input);
    Datain_tdata <= input;
    end loop;

    Datain_tvalid <= '0';
    file_close(filein);

    report "End of inputs.";
    wait;
end process;

data_output: process
file fileout            : text;
variable output_line    : LINE;
begin

  wait until rising_edge(aclk);
  file_open(fileout,"C:\..\RC_processed_Radarsat1_complete.txt", write_mode);

  while (Dataout_tvalid = '1') loop
    write(output_line,to_integer(signed(Dataout(79 downto 40))));
    write(output_line,' ');
    write(output_line,to_integer(signed(Dataout(39 downto 0))));
    writeline(fileout,output_line);
    wait until rising_edge(aclk);
  end loop;
end process;

data_FFT: process
file fileout           : text;
variable output_line   : LINE;
begin
  wait until rising_edge(aclk);
  file_open(fileout,"C:\..\RC_processed_Radarsat1_complete_FFT_output.txt", write_mode);

  while (toMF_tvalid = '1') loop
    write(output_line,to_integer(signed(toMF_tdata(47 downto 24))));
    write(output_line,' ');
    write(output_line,to_integer(signed(toMF_tdata(23 downto 0))));
    writeline(fileout,output_line);
    wait until rising_edge(aclk);
  end loop;
end process;

data_MF: process
file fileout           : text;
variable output_line   : LINE;
begin
  wait until rising_edge(aclk);
  file_open(fileout,"C:\..\RC_processed_Radarsat1_complete_MF_output.txt", write_mode);

  while (toIFFT_tvalid = '1') loop
    write(output_line,to_integer(signed(toIFFT_tdata(47 downto 24))));
    write(output_line,' ');
```

```
      write(output_line,to_integer(signed(toIFFT_tdata(23 downto 0))));
      writeline(fileout,output_line);
      wait until rising_edge(aclk);
    end loop;
  end process;
end architecture;
```

B.2 Complex Multiplier

```
entity Range_filter is
  port (
    aclk   : in  std_logic;
    reset  : in  std_logic;
    tvalid : in  std_logic;
    tlast  : in  std_logic;
    Datain : in  std_logic_vector(47 downto 0);
    Ref    : in  std_logic_vector(31 downto 0);
    isvalid: out std_logic;
    islast : out std_logic;
    Dataout: out std_logic_vector(47 downto 0)
  );
end entity;

architecture rtl of Range_filter is
------------------------------------------------------------
-- Initiation of real and imag values
------------------------------------------------------------

signal In_real  : signed(23 downto 0);
signal In_imag  : signed(23 downto 0);
signal Ref_real : signed(15 downto 0);
signal Ref_imag : signed(15 downto 0);

------------------------------------------------------------
-- Initiation of the temporary multiplied values
------------------------------------------------------------

signal tmp_real1 : signed(39 downto 0);
signal tmp_real2 : signed(39 downto 0);
signal tmp_real  : signed(39 downto 0);

signal tmp_imag1 : signed(39 downto 0);
signal tmp_imag2 : signed(39 downto 0);
signal tmp_imag  : signed(39 downto 0);

begin
  ------------------------------------------------------------
  -- Arranging the correct values
  ------------------------------------------------------------

  In_real  <= signed(Datain(47 downto 24));
  In_imag  <= signed(Datain(23 downto 0));
  Ref_real <= signed(Ref(31 downto 16));
  Ref_imag <= signed(Ref(15 downto 0));

  process(aclk,reset,tvalid) is
  begin
    if (reset = '1') then
      isvalid   <= '0';
      islast    <= '0';
```

```
      Dataout   <= (others => '0');
      tmp_real  <= (others => '0');
      tmp_real1 <= (others => '0');
      tmp_real2 <= (others => '0');
      tmp_imag  <= (others => '0');
      tmp_imag1 <= (others => '0');
      tmp_imag2 <= (others => '0');

    elsif (rising_edge(aclk)) then
      isvalid <= tvalid; -- provide 1 clk pulse delay
      islast  <= tlast;

      if (tvalid = '1') then
        tmp_real1 <= In_real * Ref_real;
        tmp_real2 <= In_imag * Ref_imag;
        tmp_real  <= tmp_real1 - tmp_real2;

        tmp_imag1 <= In_real * Ref_imag;
        tmp_imag2 <= In_imag * Ref_real;
        tmp_imag  <= tmp_imag1 + tmp_imag2;
      end if;
    end if;
    Sk <= std_logic_vector(tmp_real(39 downto 16)) & std_logic_vector(tmp_imag(39 downto 16));
  end process;
end architecture;
```

B.3 Range Pulse Replica

```
entity BRAM2048x32 is
  port (
    aclk     : in  std_logic;
    en       : in  std_logic;
    address  : in  std_logic_vector(9 downto 0);
    do       : out std_logic_vector(31 downto 0)
  );
end entity;

architecture rtl of BRAM2048x32 is
type memory is array(0 to 2047) of std_logic_vector(31 downto 0);
signal replica : memory;

begin
  process(aclk,en,address) is
  begin
    if (rising_edge(aclk)) then
      if (en = '1') then
        do <= replica(to_integer(unsigned(address)));
      else
        do <= (others => '0');
      end if;
    end if;
  end process;

  replica(0) <= "00111111011001010000100011000110";
  ...
  replica(2047) <= "11010010101100010010110100110011";
end architecture;
```

B.4 Testbench FFT

```
architecture

...

begin
  ------------------------------------------------------------------------
  -- Instantiate the DUT
  ------------------------------------------------------------------------

  dut_fft : entity work.fft_2048
    port map (
      aclk                        => aclk,
      s_axis_config_tvalid        => s_axis_config_tvalid,
      s_axis_config_tready        => s_axis_config_tready,
      s_axis_config_tdata         => s_axis_config_tdata,
      s_axis_data_tvalid          => s_axis_data_tvalid,
      s_axis_data_tready          => s_axis_data_tready,
      s_axis_data_tdata           => s_axis_data_tdata,
      s_axis_data_tlast           => s_axis_data_tlast,
      m_axis_data_tvalid          => m_axis_data_tvalid,
      m_axis_data_tdata           => m_axis_data_tdata,
      m_axis_data_tuser           => m_axis_data_tuser,
      m_axis_data_tlast           => m_axis_data_tlast,
      event_frame_started         => event_frame_started,
      event_tlast_unexpected      => event_tlast_unexpected,
      event_tlast_missing         => event_tlast_missing,
      event_data_in_channel_halt  => event_data_in_channel_halt
      );

  ------------------------------------------------------------------------
  -- Generate clock
  ------------------------------------------------------------------------

  clock_gen : process
  begin
    aclk <= '0';
    wait for CLOCK_PERIOD;
    loop
      aclk <= '0';
      wait for CLOCK_PERIOD/2;
      aclk <= '1';
      wait for CLOCK_PERIOD/2;
    end loop;
  end process clock_gen;

  ------------------------------------------------------------------------
  -- Process which controls the last frame signal to the FFT system
  ------------------------------------------------------------------------

  signal_tlast: process
  variable index : integer;
  begin
    index := 0;

    wait until rising_edge(aclk);
    while index < 2048 loop
      wait until rising_edge(aclk);
      index := index + 1;

      if (index >= 2048) then
        s_axis_data_tlast <= '1';
        index := 0;
```

```
    else
      s_axis_data_tlast <= '0';
    end if;
  end loop;
end process;

-- Function that reads the file 'Data_Radarsat1.txt' and extracts
-- 1 and 1 sample at a time. If FFT informs the system that it is busy,
-- the system will hold and wait to signal goes low.

------------------------------------------------------------------------
-- Extract data from file 'Data_Radarsat1.txt'
------------------------------------------------------------------------

data_stimuli : process
file filein          : text;
  variable input_line  : LINE;
  variable input       : std_logic_vector(15 downto 0);
  variable index_row   : integer range 0 to 2047;
  variable index_col   : integer range 0 to 2047;
begin
  s_axis_data_tvalid <= '1';
    wait until rising_edge(aclk);
  file_open(filein, "C:\Users\gbyberg\Documents\Wavemill\Data_Radarsat1.txt", read_mode);

  while(not(endfile(filein))) loop
    s_axis_data_tvalid <= '1';
    wait until rising_edge(aclk);

    readline(filein,input_line);
    read(input_line,input);
    s_axis_data_tdata <= input;
    end loop;

    s_axis_data_tvalid <= '0';
    file_close(filein);

    report "End of inputs.";
    wait;
end process;

data_output: process
variable we         : std_logic := '0';
variable open_file  : std_logic := '0';
variable close_file : std_logic := '0';
file fileout            : text;
variable output_line    : LINE;
variable file_is_open   : std_logic;
variable file_is_closed : std_logic;
begin
wait until rising_edge(aclk);
file_open(fileout,"C:\Users\gbyberg\Documents\Wavemill\FFT_processed_Radarsat1.txt", write_
--report "File has been opened now";
while (m_axis_data_tvalid = '1') loop

  write(output_line,to_integer(signed(m_axis_data_tdata(47 downto 24))));
  write(output_line,' ');
  write(output_line,to_integer(signed(m_axis_data_tdata(23 downto 0))));
  writeline(fileout,output_line);
  --report "Inside the loop now.";
  --wait;
  wait until rising_edge(aclk);
```

```
  end loop;

  --file_close(fileout);
  --report "The file has been closed";
  end process;
end tb;
```

B.5 Phase Compensation

```
entity BRAM4096x16 is
  port (
    aclk     : in  std_logic;
    en       : in  std_logic;
    address  : in  std_logic_vector(47 downto 0);
    do       : out std_logic_vector(15 downto 0)
  );
end entity;

architecture rtl of BRAM4096x16 is
type memory is array(0 to 4095) of std_logic_vector(15 downto 0);
signal phase_out : memory;
signal true_addr : std_logic_vector(11 downto 0);

begin
  true_addr <= address(47 downto 42) & address(23 downto 18);

  process(aclk,en,true_addr) is
  begin
    if (rising_edge(aclk)) then
      if (en = '1') then
        do <= phase_out(to_integer(unsigned(true_addr)));
      else
        do <= (others => '0');
      end if;
    end if;
  end process;

  phase_out(0)  <= "0000000000000000";
  ...
  phase_out(4095) <= "0011001001000011";
end architecture;
```